Unicode Functions (OS/2 Warp)

Second Edition (October 1997)

This document describes the OS/2 Warp Universal Language Support (ULS) functions. These functions provide APIs and data types to support internationalization of applications.

This document is broken up into four major sections:

- **Locale and Character Classification Functions**
- **Codepage Conversion Functions**
- **ULS Data Types**
- **Notices**

Table of Contents

Universal Language Support Functions

Internationalized applications are required to operate in a variety of environments based on some territory, language, and/or cultural definition. These environments are identified by a *locale*, an object which encapsulates culturally specific information. The locale identifies the culture, language, and territory that it supports.

UniCompleteUserLocale

UniCompleteUserLocale is used to finish a locale modification. This API is called after one or more UniSetLocaleItem calls to cause the new user defined locale file to be saved.

Format

```
#include <unidef.h>
```

int UniCompleteUserLocale
        (void)

Parameters

None required.

Returns

return value (int) - returns
    UniCompleteUserLocale returns one of the following values:

ULS_SUCCESS
    Successful completion; overridden items have been written to a file.

ERROR_OPEN_FAILED
    DosOpen failed to open the locale file.

ERROR_ACCESS_DENIED
    DosWrite failed due to denied access.

ULS_NOMEMORY
    Insufficient memory to create a buffer for writing the new locale.

Remarks

UniCompleteUserLocale is used to complete the process of defining a new locale or modifying an existing locale. An application will use the UniQueryLocale API's and UniSetUserLocaleItem API to take an existing locale definition and customize that definition to form a new locale. When the customization process is complete, the UniCompleteUserLocale API is invoked to save the results as a new locale.

The result of calling this API is that the locale is saved to disk as a new user locale or changes to an existing locale are saved to disk to represent the newly created locale.

Related Functions

- UniDeleteUserLocale
- UniMakeUserLocale

Example

```
This example shows how to complete a user locale after modifying
one or more locale items.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;

/* Array containing user locales */
UniChar     *uniUsrLocales;
int          rc = ULS_SUCCESS;
         /*************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                               */
         /*************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                   (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
```

```
        }

        /* allocate space for the user defined locales */
        uniUsrLocales = (UniChar *) malloc(4096);

        /* Query the list of user defined locales available to modify */
        rc = UniQueryLocaleList(UNI_USER_LOCALES, uniUsrLocales, 2048);

        if (rc != ULS_SUCCESS) {
          printf("UniQueryLocaleList error: return code = %u\n", rc);
          return 1;
        }

        .
        .
        .
        /* Change locale definition by calling UniSetUserLocaleItem to make
           locale item changes.
        */
        .
        .
        .


        /* Write the current set of user locales to disk */
        rc = UniCompleteUserLocale();
        if (rc != ULS_SUCCESS) {
          printf("UniCompleteUserLocale error: return code = %u\n", rc);
          return 1;
        }
        return ULS_SUCCESS;


}
```

UniCreateAttrObject

UniCreateAttrObject creates an attribute object that is used to determine character classifications.

Format

```
#include <unidef.h>
```

int UniCreateAttrObject
(const LocaleObject locale_object, const UniChar *AttrName, AttrObject *attr_object)

Parameters

locale_object (const LocaleObject)
Locale object created by a call to UniCreateLocaleObject() or NULL.

AttrName (const UniChar *)
A UniChar string that specifies the attribute names for which an attribute object should be created. Multiple attribute names are specified as a string of separate names.

attr_object (AttrObject *)
: An address that will receive a pointer to an attribute object upon successful completion of UniCreateAttrObject.

Returns

return value (int) - returns
: UniCreateLocaleObject returns one of the following values:
: **ULS_SUCCESS**
  : Successful completion; attr_object points to a valid attribute object.
: **ULS_UNSUPPORTED**
  : The attribute name specified in AttrName is not supported by the locale_object.
: **ULS_NOMEMORY**
  : Insufficient memory to create the attribute object.

Remarks

UniCreateAttrObject allocates resources associated with an attribute defined in the LC_CTYPE category of the locale indicated by the locale_object argument.

The locale_object argument specifies a locale object handle returned by UniCreateLocaleObject. It should not be a NULL pointer.

The AttrName argument specifies the attribute names for which an attribute object handle should be created. Multiple attribute names are specified as a string of space-separated names.

When UniCreateAttrObject completes without errors, the attr_object argument specifies a valid pointer to an attribute object.

The attribute object pointer should be used in all subsequent calls to the UniQueryCharAttr. If the function result is other than ULS_SUCCESS, the contents of the area pointed to by attr_object are undefined.

The following attribute names are the base POSIX attributes. All attribute names which can be specified in UniQueryCharAttr are allowed. Those attributes which start with underscore (_) or hash (#) may not be combined with other attributes.

alnum
: True when alpha or digit is true.

alpha
: True when upper or lower is true, or when none of cntrl, digit, punct, or space is true.

blank
: True for the characters space and horizontal tab.

cntrl
: True for any control character; the following attributes must be false: upper, lower, alpha, digit, xdigit, graph, print, and punct.

digit
: True for the digits 0, 1, 2 3, 4, 5, 6, 7, 8, and 9.

graph
: True for any character with the print attribute, except the space

character
: (Code element 0x0020).

lower
: True for any character that is a lowercase letter and none of cntrl, digit, punct, or space is true.

print
: True for upper, lower, alpha, digit, xdigit, punct, or any printing character including the space character (code element 0x0020).

punct
: True for any printing character that is neither the space character (code element 0x0020) nor a character for which alnum is true.

space
: True for any character that corresponds to a standard white-space character or is one of the set of white-space characters in the locale as indicated by the locale_object argument for which alnum is false. The standard white-space characters are the following: space, form feed, newline, carriage return, horizontal tab, and vertical tab.

upper
: True for any character that is an uppercase letter and none of cntrl, digit, punct, or space is true.

xdigit
: true for 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E F, a, b, c, d, e, and f.

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType

Example

```
This example shows how to create and use a character attribute object.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;

AttrObject   attr_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      uni_char = L'a';    /* Unicode lowercase Latin letter a */
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Create an alphabetic attribute object */
         rc = UniCreateAttrObject(locale_object,
                                  (UniChar *)L"alpha", &attr_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateAttrObject error: return code = %u\n", rc);
           return 1;
         }
         /* Make call to determine if character is alphabetic */
         result = UniQueryCharAttr(attr_object, uni_char);
         if (result)
           printf("UniChar character %04X is alphabetic\n", uni_char);
         else
           printf("UniChar character %04X is not alphabetic\n", uni_char);
         return ULS_SUCCESS;

}
```

UniCreateLocaleObject

UniCreateLocaleObject creates a locale object.

Format

```
#include <unidef.h>
```

int UniCreateLocaleObject
    (int LocaleSpecType, const void *LocaleSpec, LocaleObject *locale_object)

Parameters

LocaleSpecType (int)
  Identifies the type of value in the **LocaleSpec** argument.

  The constant names of the values of **LocaleSpecType** are defined in the header **unidef.h**:

  UNI_TOKEN_POINTER
    LocaleSpec points to a locale token.
  UNI_MBS_STRING_POINTER
    LocaleSpec points to a multibyte character string.
  UNI_UCS_STRING_POINTER
    LocaleSpec points to a UCS character string.

LocaleSpec (const void *)
  The **LocaleSpec** argument points to either a character string or a locale token, as indicated by the value of the **LocaleSpecType** argument.

locale_object (LocaleObject *)
  An address that will receive a pointer to a locale object upon successful completion of UniCreateLocaleObject.

Returns

return value (int) - returns

UniCreateLocaleObject returns one of the following values:

ULS_SUCCESS
The specified locale is supported and a valid locale object was created.

ULS_UNSUPPORTED
The specified locale is not supported; the locale object pointer points to undefined data.

ULS_NOMEMORY
There is insufficient memory to create the requested locale or the default locale; the locale object pointer points to undefined data.

ULS_INVALID
An invalid locale specification string or token was passed; the locale object pointer points to undefined data.

Remarks

UniCreateLocaleObject creates a locale object for the locale specified by **LocaleSpec**. The object created is an opaque object containing all the data and methods necessary to perform the language-sensitive operations or functions that accept an argument of type **LocaleObject**. If the function is successful, all categories of the locale object are created and initialized.

When the **LocaleSpec** argument is a pointer to a character string (UCS character string or multibyte character string), it identifies the name of the locale to be initialized. The locale name is used to locate physical resources associated with this locale. The locale name **UNIV** is reserved and refers to the definitions that provide default behavior for functions.

When the **LocaleSpec** argument is a NULL pointer (without regard to the value of the **LocaleSpecType** argument), UniCreateLocaleObject creates a locale object for the **UNIV** locale.

When the **LocaleSpec** argument points to a locale token value as indicated by the value of the **LocaleSpecType** argument, the token identifies the locale to be initialized.

When the **LocaleSpec** argument is an empty multibyte or UCS character string, UniCreateLocaleObject creates a locale object based upon the settings of the locale environment variables.

Locale Environment Variables by Precedence and Usage

Catgeory	Precedence	Usage
LC_ALL	Highest	Setting LC_ALL takes precedence over any other locale environment variable.
LC_COLLATE	Equal precedence	Specifies collation (sorting) rules.
LC_CTYPE	Equal precedence	Specifies character classification and case conversion.
LC_MESSAGES	Equal precedence	Specifies the values for affirmative and negative answers, and the language for displayed messages.
LC_MONETARY	Equal precedence	Specifies monetary formats and currency symbol.

LC_NUMERIC	Equal precedence	Specifies decimal formats.
LC_TIME	Equal precedence	Specifies date and time formats.
LANG	Lowest	Setting LANG takes precedence over any undefined locale environment variable. This may be used in conjunction with LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, and LC_TIME.

If the specified locale is valid and supported, UniCreateLocaleObject allocates memory for the new object and returns the address of the created locale object in the area pointed to by **locale_object**. It is the application's responsibility to free this memory with a call to UniFreeLocaleObject when the object is no longer needed. If the function fails for any reason, the contents of the area pointed to by **locale_object** are undefined.

The locale token provides a shorthand notation for specifying a locale. The format of the locale token is as returned by a call to UniLocaleStrToToken. The format is defined as an unsigned integer of four octets.

Examples of typical usage:

The locale environment variables are set as follows:

LANG=de_DE
LC_MONETARY=en_US

The **LocaleSpec** argument is an empty multibyte or UCS character string.

```
This example creates a locale object with all categories set to de_DE except for
LC_MONETARY which has the value of en_US.
```

The locale environment variables are set as follows:

LANG=fr_FR

The **LocaleSpec** argument is an empty multibyte or UCS character string.

```
This example creates a locale object with all categories set to fr_FR.
```

The locale environment variables are set as follows:

LC_ALL=it_IT
LANG=fr_FR

The **LocaleSpec** argument is an empty multibyte or UCS character string.

```
This example creates a locale object with all categories set to it_IT.
```

Related Functions

- UniFreeLocaleObject

Example

```
This example shows how to create a locale object.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;

int          rc = ULS_SUCCESS;
         /*******************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*******************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;

}
```

UniCreateTransformObject

UniCreateTransformObject creates a string transform object.

Format

```
#include <unidef.h>
```

int UniCreateTransformObject
(const LocaleObject locale_object, const UniChar *xtype, XformObject *xform_object)

Parameters

locale_object (const LocaleObject)
A locale object created by UniCreateLocaleObject or NULL.

xtype (const UniChar *)
A UniChar string identifying the transform type.

xform_object (XformObject *)
An address that will receive a pointer to an Xform Object upon successful completion of UniCreateTransformObject.

Returns

return value (int) - returns

UniCreateTransformObject returns one of the following:

ULS_SUCCESS
: No errors; the xform_object argument points to a valid transformation object.

ULS_UNSUPPORTED
: The transformation name type specified by the xtype argument is not supported for locale_object.

Remarks

UniCreateTransformObject obtains a transformation object for a transformation type as defined in the locale indicated by the locale_object argument. The function returns a transformation object that can be used as an argument in UniTransformStr.

The following transformation types are defined in all locales:

lower
: Transform to lowercase characters. A character that does not have a lowercase form is returned as itself.

upper
: Transform to uppercase characters. A character that does not have an uppercase form is returned as itself.

compose
: Transform to fully composed form for combined characters.

decompose
: Transform to a string of decomposed characters where multiple characters are used to represent base and diacritics.

hiragana
: Transform so that Japanese phonetic characters are in hiragana

katakana
: Transform so that Japanese phonetic characters are in full size katakana

kana
: Transform so that Japanese phonetic characters are in half size katakana

In addition to the above transformation-type names, other transformation-type names in the locale (including user-defined transformation-type names) may be passed to UniCreateTransformObject through the xtype argument. To obtain a successful return, the transformation-type name must be defined in locale_object.

When UniCreateTransformObject completes without errors, the xform_object argument value specifies a valid pointer to a transformation object. The transformation object should be used in all subsequent calls to UniTransformStr. If the function result is other than ULS_SUCCESS, the contents of the area pointed to by **xform_object** are undefined.

Related Functions

- UniFreeTransformObject

Example

```
This example shows how to create and use a transform object.
#include <stdio.h>

#include <unidef.h>
```

```
int main(void) {
LocaleObject locale_object = NULL;

XformObject  xform_object = NULL;
int          rc = ULS_SUCCESS;
int          in_unistr_elem = 0;
int          out_unistr_elem = 10;
UniChar      *pin_unistr = (UniChar *)L"os2";
UniChar      out_unistr[10];
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Create an upper case transform object */
         rc = UniCreateTransformObject(locale_object,
                                       (UniChar *)L"upper", &xform_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateTransformObject error: return code = %u\n", rc);
           return 1;
         }
         /* Calculate the number of elements to transform */
         in_unistr_elem = UniStrlen (pin_unistr) + 1;
         /* Make call to transform input string to uppercase */
         rc = UniTransformStr(xform_object, pin_unistr,
                              &in_unistr_elem, out_unistr,
                              &out_unistr_elem);
         if (rc != ULS_SUCCESS) {
           printf("UniTransformStr error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;

}
```

UniDeleteUserLocale

UniDeleteUserLocale is used to delete a locale created by a user.

Format

```
#include <unidef.h>
```

int UniDeleteUserLocale
    (UniChar * locale)

Parameters

locale (UniChar *)
    A pointer to a UniChar string which defines the name of the locale.

Returns

return value (int) - returns

UniDeleteUserLocale returns on of the following:

ULS_SUCCESS
: Successful completion; user locale deleted from disk.

ULS_NOMATCH
: The requested locale cannot be found.

ULS_INVALID
: The locale being deleted is not a user defined locale.

Remarks

UniCompleteDeleteLocale is used to remove a previously defined user locale. The locale must have been previously created as a user locale. This is accomplished by using the UniCompleteUserLocale API.

Related Functions

- UniCompleteUserLocale
- UniMakeUserLocale

Example

```
This example shows how to delete a user locale once it is no longer
needed by the user.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;

/* Array containing user locales */
UniChar       *uniUsrLocales;
UniChar       uniLocaleName[MAX_LOCALE_NAME_LENGTH];
int           rc = ULS_SUCCESS;
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }

         .
         .
         .
         /*
             Identify the locale to be deleted - making sure the name is in
             Unicode.
         */
         .
         .
         .
         .
         /* Delete a user locale from the disk */
         rc = UniDeleteUserLocale(uniLocaleName);
         if (rc != ULS_SUCCESS) {
           printf("UniDeleteUserLocale error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;
```

}

UniFreeAttrObject

UniFreeAttrObject frees the character attribute object.

Format

```
#include <unidef.h>
```

int UniFreeAttrObject
(AttrObject attr_object)

Parameters

attr_object (AttrObject)
An attribute object to be freed. The attribute object must have been created by a call to UniCreateAttrObject.

Returns

return value (int) - returns

UniFreeAttrObject returns one of the following values:

ULS_SUCCESS
All resources associated with the attribute object specified by the attr_object argument have been successfully deallocated.

ULS_BADOBJ
The attribute object specified by attr_object is not a valid attribute object.

Remarks

UniFreeAttrObject releases all resources associated with the character attribute object allocated by UniCreateAttrObject.

The attr_object argument specifies a previously allocated attribute object.

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType

Example

```
This example shows how to create and free a character attribute object.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;

AttrObject   attr_object = NULL;

int          rc = ULS_SUCCESS;
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Create an alphabetic attribute object */
         rc = UniCreateAttrObject(locale_object,
                                  (UniChar *)L"alpha", &attr_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateAttrObject error: return code = %u\n", rc);
           return 1;
         }
         /* Free the character attribute object */
         rc = UniFreeAttrObject(attr_object);
         if (rc != ULS_SUCCESS) {
           printf("UniFreeAttrObject error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;

}
```

UniFreeLocaleInfo

UniFreeLocaleInfo frees a locale information structure created by UniQueryLocaleInfo.

Format

```
#include <unidef.h>
```

int UniFreeLocaleInfo
    (struct UniLconv *UniLconv_addr)

Parameters

UniLconv_addr (struct UniLconv *)
    A locale information structure created by a call to UniQueryLocaleInfo.

Returns

return value (int) - returns
    UniFreeLocaleInfo returns one of the following values:
    ULS_SUCCESS
        The UniLconv structure and associated memory were successfully freed.
    ULS_BADOBJ
        The UniLconv_addr is not a valid structure.

Related Functions

- UniQueryLocaleInfo

Example

```
This example shows how to create and free a locale information structure.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject      locale_object = NULL;
struct UniLconv   *puni_lconv = NULL;

int               rc = ULS_SUCCESS;
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Retrieve locale information */
         rc = UniQueryLocaleInfo(locale_object, &puni_lconv);
         if (rc != ULS_SUCCESS) {
           printf("UniQueryLocaleInfo error: return code = %u\n", rc);
           return 1;
         }
         printf("Monetary decimal point is %ls\n", puni_lconv->mon_decimal_point);
         /* Free the locale information structure */
         rc = UniFreeLocaleInfo(puni_lconv);
         if (rc != ULS_SUCCESS) {
           printf("UniFreeLocaleInfo error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;

}
```

UniFreeLocaleObject

UniFreeLocaleObject frees a locale object that was created by UniCreateLocaleObject.

Format

```
#include <unidef.h>
```

int UniFreeLocaleObject
(LocaleObject locale_object)

Parameters

locale_object (LocaleObject)
The Locale Object to be freed. locale_object must have been created by a call to UniCreateLocaleObject.

Returns

return value (int) - returns
UniQueryLocaleObject returns one of the following values:
ULS_SUCCESS
A valid locale specification for the supplied locale object is returned.
ULS_BADOBJ
Invalid locale object specified.

Remarks

The UniFreeLocaleObject function destroys the locale object identified by locale_object and frees any memory associated with it.

Related Functions

- UniQueryLocaleObject

Example

```
This example shows how to create and free a locale object.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject      locale_object = NULL;
int               rc = ULS_SUCCESS;
         /* Create a locale object for French in Canada */
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"fr_CA", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
```

```
        }
        /* Free the locale object that was just created */
        rc = UniFreeLocaleObject(locale_object);
        if (rc != ULS_SUCCESS) {
          printf("UniFreeLocaleObject error: return code = %u\n", rc);
          return 1;
        }
        return ULS_SUCCESS;

}
```

UniFreeMem

UniFreeMem frees memory allocated by UniQueryLocaleObject.

Format

```
#include <unidef.h>
```

int UniFreeMem
  (void *memory)

Parameters

memory (void *) - in/out
  A pointer to the memory to be freed.

Returns

Returns (int) - returns
  UniFreeMem returns one of the following values:
  ULS_SUCCESS
    Indicates success.
  ULS_BADOBJ
    Invalid pointer in memory.

Remarks

UniFreeMem frees memory allocated by ULS functions. For example, the memory allocated for the locale_name parameter of UniQueryLocaleObject should be freed using UniFreeMem.

Example

```
This example shows how to free memory allocated by a ULS function.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject       locale_object = NULL;
int                rc = ULS_SUCCESS;
char               *locale_name;
         /* Create a locale object for French in Canada */
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"fr_CA", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Determine the locale name for the LC_MESSAGES category */
         rc = UniQueryLocaleObject(locale_object, LC_MESSAGES,
                                    UNI_MBS_STRING_POINTER,
                                    (void **)&locale_name);
         if (rc != ULS_SUCCESS) {
           printf("UniQueryLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Free the memory allocated by UniQueryLocaleObject */
         rc = UniFreeMem((void **)locale_name);
         if (rc != ULS_SUCCESS) {
           printf("UniFreeMemObject error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;

}
```

UniFreeTransformObject

UniFreeTransformObject frees a string transformation object.

Format

```
#include <unidef.h>
```

int UniFreeTransformObject
(XformObject xform_object)

Parameters

xform_object (XformObject)
The transform object to be freed. The transform object must have been created by a call to UniCreateTransformObject.

Returns

return value (int) - returns

UniFreeTransformObject returns one of the following:

ULS_SUCCESS
Specifies that all resources associated with the transformation object specified by the xform_object argument have been successfully deallocated.

Remarks

UniFreeTransformObject releases all resources associated with a transformation object previously obtained by UniCreateTransformObject.

Related Functions

- UniCreateTransformObject

Example

```
This example shows how to create and free a transform object.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;

XformObject  xform_object = NULL;

int          rc = ULS_SUCCESS;
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                   (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Create an upper case transform object */
         rc = UniCreateTransformObject(locale_object,
                                      (UniChar *)L"lower", &xform_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateTransformObject error: return code = %u\n", rc);
           return 1;
         }
         /* Free the transform object created by UniCreateTransformObject */
         rc = UniFreeTransformObject(xform_object);
         if (rc != ULS_SUCCESS) {
           printf("UniFreeTransformObject error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;

}
```

UniLocaleStrToToken

UniLocaleStrToToken converts a locale specification string to a token.

Format

```
#include <unidef.h>
```

int UniLocaleStrToToken
(int LocaleStringType, const void *locale_string, LocaleToken *locale_token)

Parameters

LocaleStringType (int)
Informs UniLocaleStrToToken of the type of string being passed in the locale_string variable.

The LocaleStringType argument can take any of the following values, which are constants defined in the header unidef.h:

UNI_MBS_STRING_POINTER
Requests that a multibyte string pointer is held in locale_string.
UNI_UCS_STRING_POINTER
Requests that a UCS string pointer is held in locale_string.

locale_string (const void *)
The locale specification string.

locale_token (LocaleToken *)
An address that will receive a pointer to the newly created token corresponding to locale_string.

Returns

return value (int) - returns
UniLocaleStrToToken returns one of the following values:
ULS_SUCCESS
A valid locale token for the supplied locale object is returned.
ULS_OTHER
The C locale is because LOCALE.DLL cound not be found.
ULS_UNSUPPORTED
The locale name is valid but the locale cound not be found.

Remarks

UniLocaleStrToToken accepts, as an argument, a locale string qualified by the value of the **LocaleStringType** argument. It returns a locale token pointed to by **locale_token** if such a token exists for that locale string. UniLocaleStrToToken allocates memory to hold the locale token value. If no locale token exists for the supplied locale string, the value returned in **locale_token** is undefined.

Related Functions

- UniLocaleTokenToStr

Example

```
This example shows how to convert a locale specification string to a token.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar       *locale_string = L"de_DE";  /*German in Germany locale string */
LocaleToken   locale_token;

int           rc = ULS_SUCCESS;
         rc = UniLocaleStrToToken(UNI_UCS_STRING_POINTER,
                                 (void *)locale_string,
                                 &locale_token);
         if (rc != ULS_SUCCESS) {
           printf("UniLocaleStrToToken error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;

}
```

UniLocaleTokenToStr

UniLocaleTokenToStr converts a locale token to a locale specification string.

Format

```
#include <unidef.h>
```

int UniLocaleTokenToStr
(const LocaleToken locale_token, int LocaleStringType, void **locale_string)

Parameters

locale_token (const LocaleToken)
A token identifying a locale.

LocaleStringType (int)
The LocaleStringType argument can take any of the following values, which are constants defined in the header unidef.h:

UNI_MBS_STRING_POINTER
Requests that a multibyte string pointer be returned.

UNI_UCS_STRING_POINTER
Requests that a UCS string pointer be returned.

locale_string (void **)

An address of a pointer variable locale_string that will contain the locale specification string corresponding to locale_token.

Returns

return value (int) - returns

The UniLocaleTokenToStr function returns one of the following values:

ULS_SUCCESS
: A valid locale specification for the supplied locale object is returned.

ULS_INVALID
: The locale token supplied could not be matched to a locale string.

ULS_NOMEMORY
: There is insufficient memory to store the locale string.

Remarks

The UniLocaleTokenToStr() function accepts as an argument a locale token in **locale_token** and returns a pointer to a locale string in **locale_string** qualified by the LocaleStringType argument. The UniLocaleTokenToStr() function allocates memory to hold the locale string value. It is the application's responsibility to free the memory using UniFreeMem() when the locale string value is no longer needed. If no locale string can be generated for the supplied locale token, the value returned in **locale_string** is undefined.

Related Functions

- UniLocaleStrToToken

Example

```
This example shows how to convert a locale token to a locale specification string.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar       *locale_string1 = L"de_DE";  /*German in Germany locale string */
UniChar       *locale_string2;

LocaleToken   locale_token;

int           rc = ULS_SUCCESS;
         rc = UniLocaleStrToToken(UNI_UCS_STRING_POINTER,
                                  (void *)locale_string1,
                                  &locale_token);
         if (rc != ULS_SUCCESS) {
           printf("UniLocaleStrToToken error: return code = %u\n", rc);
           return 1;
         }
```

/* Convert the token to a locale string */

```
         rc = UniLocaleTokenToStr(locale_token,
                                  UNI_UCS_STRING_POINTER,
                                  (void **)&locale_string2);
         if (rc != ULS_SUCCESS) {
           printf("UniLocaleTokenToStr error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;
```

}

UniMakeUserLocale

UniMakeUserLocale creates a user locale from a base system locale.

Format

```
#include <unidef.h>
```

int UniMakeUserLocale
(UniChar * newName, UniChar * baseName)

Parameters

newName (UniChar *)
The name of the new locale.

baseName (UniChar *)
The name of the locale to base the new locale after.

Returns

return value (int) - returns
UniMakeUserLocale returns one of the following values:

ULS_SUCCESS
The user locale has been created.
ULS_NOMATCH
The base system locale does not exist.
ULS_INVALID
The name supplied contains an illegal character, is too long or redefines a base system locale.
ULS_NOOP
The name supplied currently exists as a locale name.
ULS_NOMEMORY
Cannot allocate memory for the new locale.

Remarks

The names for the new locale and the base system locale must be ASCII-7 chars and at most eight characters, in length. An existing locale name must be given as the base locale name.

If the user locale already exists, a ULS_NOOP return code will be given, therefore, this API can always be called before making an update to ensure that the user locale exists.

Related Functions

- UniCompleteUserLocale

- UniDeleteUserLocale

Example

```
This example shows how to make a user locale.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject   locale_object = NULL;

int            rc = ULS_SUCCESS;
UniChar        *plocaleName;
UniChar        *puniSysLocale;
    /* Create current default locale object for this process */
    rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                         (UniChar *)L"",
                         &locale_object);
    if(rc) {
      printf("UniCreateLocaleObject error: return code = %u\n", rc);
      return 1;
    }
    /* Query the name of the default locale object */
    rc = UniQueryLocaleObject(locale_object,
                         LC_ALL,
                         UNI_UCS_STRING_POINTER,
                         (void**)&plocaleName);
    if(rc) {
      printf("UniQueryLocaleObject error: return code = %u\n", rc);
      return 1;
    }
    /* Get the locale name from the locale object string */
    puniSysLocale = UniStrtok(plocaleName, (UniChar *)L" ");
    /* Make a new locale */
    rc = UniMakeUserLocale(puniSysLocale, puniSysLocale);
    if (rc) {
      printf("UniMakeUserLocale error: return code = %u\n", rc);
      return 1;
    }
    /* free the space used by the locale object string */
    UniFreeMem(plocaleName);
    if(locale_object)
        UniFreeLocaleObject(locale_object);
    return ULS_SUCCESS;

}
```

UniMapCtryToLocale

UniMapCtryToLocale converts an unsigned long country code into a locale name represented as a UniChar string that is acceptable as input to other Unicode APIs.

Format

```
#include <unidef.h>
```

int UniMapCtryToLocale
(unsigned long ulCountryCode, UniChar *ucsLocaleName, size_t n)

Parameters

ulCountryCode (unsigned long) - input
    An OS/2 country code.

ucsLocaleName (UniChar *) - output
    A buffer for placing the Unicode string.

n (size_t) - input
    Size, in characters, of the ucsLocaleName buffer. This should be at least 8 Unicode characters.

Returns

return value (int) - returns
    Error code.

UniMapCtryToLocale returns one of the following values:

ULS_SUCCESS
A valid locale name for the supplied country code is returned.
ULS_BUFFERFULL
The buffer is not large enough to hold the locale name.
ULS_INVALID
An invalid country code or buffer was specified.

Related Functions

- UniMapCpToUcsCp

Example

```
This example shows how to map a country code to a locale name.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar        ucs_locale_name[8];
size_t         num_elems = 8;
```

unsigned long country_num = 1;
LocaleObject locale_object = NULL;
int rc = ULS_SUCCESS;

```
         /*******************************************************************/
         /* Convert country number to a locale name                         */
         /*******************************************************************/
         rc = UniMapCtryToLocale(country_num, ucs_locale_name, num_elems);
         if (rc != ULS_SUCCESS) {
           printf("UniMapCtryToLocale error: return code = %u\n", rc);
           return 1;
         }
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                   ucs_locale_name, &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;
```

}

UniQueryAlnum

UniQueryAlnum queries character attributes.

Format

```
#include <unidef.h>
```

int UniQueryAlnum
    (const LocaleObject locale_object, UniChar uc)

Parameters

locale_object (const LocaleObject)
    A locale object created by UniCreateLocaleObject or NULL.

uc (UniChar)
    The UniChar character to query.

Returns

return value (int) - returns
    If the result of the test is true, the function returns 1. Otherwise, 0 is returned.

Remarks

This function provides the functionality of UniCreateAttrObject, UniQueryCharAttr, and UniFreeAttrObject as an atomic operation for the invariant attributes.

The locale may be specified as NULL to indicate default Unicode character attributes

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType

Example

```
This example shows how to query character attributes.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      uni_char = L'a';    /* Unicode lowercase Latin letter a */
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Query character attribute */
         result = UniQueryAlnum(locale_object, uni_char);
         if (result)
           printf("UniChar character %04X is alphanumeric\n", uni_char);
         else
           printf("UniChar character %04X is not alphanumeric\n", uni_char);
```

return ULS_SUCCESS;
}

UniQueryAlpha

UniQueryAlpha queries character attributes.

Format

```
#include <unidef.h>
```

int UniQueryAlpha
(const LocaleObject locale_object, UniChar uc)

Parameters

locale_object (const LocaleObject)
A locale object created by UniCreateLocaleObject or NULL.

uc (UniChar)
The UniChar character to query.

Returns

Return Value (int) - returns
  If the result of the test is true, the function returns 1. Otherwise, 0 is returned.

Remarks

This function provides the functionality of UniCreateAttrObject, UniQueryCharAttr, and UniFreeAttrObject as an atomic operation for the invariant attributes.

The locale may be specified as NULL to indicate default Unicode character attributes

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType

Example

```
This example shows how to query character attributes.
#include <stdio.h>

#include <unidef.h>
int main(void) {
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      uni_char = L'a';    /* Unicode lowercase Latin letter a */
         /* Query character attribute */
         result = UniQueryAlpha(NULL, uni_char);
         if (result)
           printf("UniChar character %04X is alphabetic\n", uni_char);
         else
           printf("UniChar character %04X is not alphabetic\n", uni_char);
```

return ULS_SUCCESS;
}

UniQueryAttr

UniQueryAttr returns the value associated with attribute name supplied by the user.

Format

```
#include <unidef.h>
```

ulong UniQueryAttr
  (UniChar * attrName)

Parameters

attrName (UniChar *)
The name of a character attribute.

Returns

Return Value (ulong) - returns
If the attribute name is known, the function returns the attribute value. Otherwise, 0 is returned.

Remarks

This function provides the numeric value for the standard attributes such as alpha, graph, and number. In addition, this function provides the numeric value for other attributes such as Hiragana, diacritic, halfwidth etc. The table below contains the valid attribute names. Valid names are all in lower case.

Attribute names that begin with a lower case letter may be ORed together.

Attribute Name and Description Table

Attr Name	**Attribute Define**	**Description of Attribute**
alnum	CT_ALNUM	Alphabetic and numeric characters
alpha	CT_ALPHA	Letters and linguistic marks
ascii	CT_ASCII	Standard ASCII character
blank	CT_BLANK	Space and Tab
cntrl	CT_CNTRL	Control and format characters
diacritic	C3_DIACRITIC	Diacritic
digit	CT_DIGIT	Digits 0 through 9
fullwidth	C3_FULLWIDTH	Full width variant
graph	CT_GRAPH	All except controls and space
halfwidth	C3_HALFWIDTH	Half width variant
hiragana	C3_HIRAGANA	Hiragana character
ideograph	C3_IDEOGRAPH	Kanji/Han character
kashida	C3_KASHIDA	Arabic tatweel (used to stretch characters)
katakana	C3_KATAKANA	Katakana character
lower	CT_LOWER	Lower case alphabetic character

nonspacing	C3_NONSPACING	Non-spacing mark
nsdiacritic	C3_NSDIACRITIC	Non-spacing diacritic
nsvowel	C3_NSVOWEL	Non-spacing vowel
number	CT_NUMBER	Integers between 0 and 9
print	CT_PRINT	Everything except control characters
punct	CT_PUNCT	Punctuation marks
space	CT_SPACE	Whitespace and line ends
symbol	CT_SYMBOL	Symbol
upper	CT_UPPER	Upper case alphabetic character
vowelmark	C3_VOWELMARK	Vowel mark
xdigit	CT_XDIGIT	Hexadecimal digits (0-9, a-f or A-F)
_apl	CHS_APL	APL character
_arabic	CHS_ARABIC	Arabic character
_arrow	CHS_ARROW	Arrow character
_bengali	CHS_BENGALI	Bengali character
_bopomofo	CHS_BOPOMOFO	Bopomofo character
_box	CHS_BOX	Box or line drawing character
_currency	CHS_CURRENCY	Currency Symbol
_cyrillic	CHS_CYRILLIC	Cyrillic character
_dash	CHS_DASH	Dash character
_dingbat	CHS_DINGBAT	Dingbat
_fraction	CHS_FRACTION	Fraction value
_greek	CHS_GREEK	Greek character
_gujarati	CHS_GUJARATI	Gujarati character
_gurmukhi	CHS_GURMUKHI	Gurmukhi character

_hanguel	CHS_HANGUEL	Hanguel character
_hebrew	CHS_HEBREW	Hebrew character
_hiragana	CHS_HIRAGANA	Hiragana character set
_katakana	CHS_KATAKANA	Katakana character set
_lao	CHS_LAO	Laotian character
_latin	CHS_LATIN	Latin character
_linesep	CHS_LINESEP	Line separator
_math	CHS_MATH	Math symbol
_punctstart	CHS_PUNCTSTART	Punctuation start
_punctend	CHS_PUNCTEND	Punctuation end
_tamil	CHS_TAMIL	Tamil character
_telegu	CHS_TELEGU	Telegu character
_thai	CHS_THAI	Thai character
_userdef	CHS_USERDEF	User defined character
#arabicnum	C2_ARABICNUMBER	Arabic numbers
#blocksep	C2_BLOCKSEPARATOR	Block separator
#commonsep	C2_COMMONSEPARATOR	Common separator
#euronum	C2_EUROPENUMBER	European number
#eurosep	C2_EUROPESEPARATOR	European separator
#euroterm	C2_EUROPETERMINATOR	European terminator
#left	C2_LEFTTORIGHT	Left to right text orientation
#mirrored	C2_MIRRORED	Symmetrical text orientation
#neutral	C2_OTHERNEUTRAL	Other neutral
#right	CT_RIGHTTOLEFT	Right to left text orientation
#whitespace	C2_WHITESPACE	Whitespace

Related Functions

- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType

Example

```
This example shows how to query character attribute values using the
character attributes.
#include <stdio.h>
#include <unidef.h>
int main(void) {
    char    name[33];
    UniChar uname[33];
    UniChar * up;
    char    * cp;
    ulong   rc;
    /*
     * Ask the user for an attribute name
     */
    printf("Enter attribute name:");
    scanf("%s", &name);
    if (strlen(name) > 32)
      return 1;
    /*
     * Convert name to unicode
     */
    cp = name;
    up = uname;
    while (*cp) {
      *up++ = (UniChar)(*cp++);
    }
    *up = 0;
    /*
     * Query the attribute and print the value
     */

    rc = UniQueryAttr(tolower(uname));
    if (rc == 0) {
       printf("UniQueryAttr error: return code = %u\n", rc);
       return 1;
    } else
       printf("%s attribute = %x\n", name, rc);
       return ULS_SUCCESS;
}
```

UniQueryBlank

UniQueryBlank queries character attributes.

Format

```
#include <unidef.h>
```

int UniQueryBlank
(const LocaleObject locale_object, UniChar uc)

Parameters

locale_object (const LocaleObject)
A locale object created by UniCreateLocaleObject or NULL.

uc (UniChar)
The UniChar character to query.

Returns

Return Value (int) - returns
If the result of the test is true, the function returns 1. Otherwise, 0 is returned.

Remarks

This function provides the functionality of UniCreateAttrObject, UniQueryCharAttr, and UniFreeAttrObject as an atomic operation for the invariant attributes.

The locale may be specified as NULL to indicate default Unicode character attributes

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType

Example

```
This example shows how to query character attributes.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      uni_char = L' ';    /* Unicode space character */
         /*******************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                  */
         /*******************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Query character attribute */
         result = UniQueryBlank(locale_object, uni_char);
```

```
        if (result)
          printf("UniChar character %04X is a blank character\n", uni_char);
        else
          printf("UniChar character %04X is not a blank character\n", uni_char);
```

return ULS_SUCCESS;
}

UniQueryChar

UniQueryChar determines if the character supplied has the attribute(s) requested.

Format

```
#include <unidef.h>
```

int UniQueryChar
(UniChar uc, ULONG attrName)

Parameters

uc (UniChar)
The Unicode character whose attribute(s) are being examined.

attrName (ULONG)
The name of the attribute being examined in the Unicode character.

Returns

Return Value (ulong) - returns
If the named attribute is true for the Unicode character supplied, the function returns a 1. Otherwise, 0 is returned.

Remarks

This function takes the attributes supplied by the caller and tests the character to determine if they are true for that Unicode character. Attribute names that have a leading _ or # character represent classes of characters. These attributes must be tested as individual attributes. The remaining attributes can be or'ed together before testing.

Related Functions

- UniQueryAttr
- UniQueryCharAttr
- UniQueryCharType

Example

```
This example shows how to query if a character has particular attributes.
#include <stdio.h>
#include <unidef.h>
int main(void) {
         int          result = 0;
         UniChar      uni_char = L'A';    /* Unicode A character */

         /* Query character for upper case and graphic attributes */
         result = UniQueryChar(uni_char, C1_UPPER || C1_GRAPH);
         if (result)
           printf("UniChar is upper case and a graphic character\n");
         else
           printf("UniChar is not upper case and a graphic character\n"_;
         /* Query character for Latin character set attribute */
         result = UniQueryChar(uni_char, CHS_LATIN);
         if (result)
           printf("UniChar is a Latin character\n");
         else
           printf("UniChar is not a Latin character\n");

         return ULS_SUCCESS;
}
```

UniQueryCharAttr

UniQueryCharAttr queries the attributes of a character.

Format

```
#include <unidef.h>
```

int UniQueryCharAttr
    (AttrObject attr_object, UniChar uc)

Parameters

attr_object (AttrObject)
    An attribute object created by UniCreateAttrObject.

uc (UniChar)
    The UniChar character whose attributes will be queried.

Returns

return value (int) - returns
    If the result of the test is true (the code element has the attribute associated with the attribute object, attr_object), UniQueryCharAttr returns the value 1.

    If the result of the test is false, UniQueryCharAttr returns the value 0.

Remarks

UniQueryCharAttr determines whether the code element uc has the attributes specified by the attribute object argument, attr_object.

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharType

Example

```
This example shows how to create and use a character attribute object.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;

AttrObject   attr_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      uni_char = L'c';    /* Unicode lowercase Latin letter c */
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                   (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Create an attribute object */
         rc = UniCreateAttrObject(locale_object,
                                 (UniChar *)L"alpha xdigit", &attr_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateAttrObject error: return code = %u\n", rc);
           return 1;
         }
         /* Make call to determine if character matches attributes */
         result = UniQueryCharAttr(attr_object, uni_char);
         if (result)
           printf("UniChar character %04X matches attributes\n", uni_char);
         else
           printf("UniChar character %04X does not match attributes\n", uni_char);
         return ULS_SUCCESS;

}
```

UniQueryCharType

UniQueryCharType is used to query the type of the character.

Format

```
#include <unidef.h>
```

UNICTYPE * UniQueryCharType
(UniChar uc)

Parameters

uc (UniChar)
The UniChar character whose type will be queried.

Returns

return value (UNICTYPE *) - returns
A pointer to a structure of type UNICTYPE Iis returned from this call.

Remarks

UniQueryCharType is designed to provide information to support both the XPG/4 character type as well as the Win32 GetCharType type. Where the function is similar, this API is designed to be a superset of the Win32 function so that the Win32 functions can be supported by masking off bits in the returned data structure. GetCharType is similar to the C library "is" functions.

The UNICTYPE structure contains character set information, information regarding Bidirectional attributes, information regarding XPG/4 attributes and information on extended attributes.

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharTypeTable

Example

```
This example shows how to query a character type.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UNICTYPE     * uct;
UniChar      uni_char = 0x3456;    /* Some random Unicode character */

     /* Query the character type */
     uct = UniQueryCharType(uni_char);

     /* Examine the returned structure to determine information about
        the character.  For example, what is its BiDi orientation and
        is the character Arabic or Hebrew? */
     if (uct->bidi==C2_RIGHTTOLEFT)
        printf("Character is presented right to left\n");
     else
        printf("Character is presented left to right\n");

     if (uct->charset==CHS_ARABIC)
        printf("Character is Arabic\n");
     else if (uct->charset==CHS_HEBREW)
        printf("Character is Hebrew\n");
     else
        printf("Character is not Arabic or Hebrew\n");

return ULS_SUCCESS;
}
```

UniQueryCharTypeTable

UniQueryCharTypeTable is used to query the type of the character.

Format

```
#include <unidef.h>
```

ULONG UniQueryCharTypeTable
(ULONG * count, UNICTYPE * * unictype)

Parameters

count (ULONG *)
: count is set to the length of the table being accessed.

unictype (UNICTYPE * *)
: unictype is set to point to a table of UNICTYPE structures.

Returns

return value (ULONG) - returns
: A ULONG equal to zero is always returned.

Remarks

UniQueryCharTypeTable is passed a pointer to a count and a pointer to a table of UNICTYPE structures. count is set to the number of entries in the UNICTYPE structure table. unictype is set to the first structure in the table.

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType

Example

```
This example shows how to query a character type table.
#include <unidef.h>
```

```
#include <stdlib.h>

    /*
     * StringBidi: Determine bidi types for each character in a string
     * Return string of bidi bits, and return value with
     * OR of all bits.
     */
    USHORT StringBidi(UniChar * instr, USHORT * charsets) {
    ULONG count;
    UNICTYPE * typetab;
    USHORT index, *pcs, out;
    int rc, i, len;

    /*
     * Get addressability to the character type table
     */
    UniQueryCharTypeTable (&count, &typetab);

    /*
     * Create an output string
     */
    len = UniStrlen(instr);
    UniQueryStringType(instr, len, charsets, CT_INDEX);

    /*
     * Replace each index with bidi flags
     */
    pcs = charsets;
    out = 0;
    for (i=0; i<len; i++) {
        index = *pcs;
        *pcs = 0;
        if (typetab[index].bidi == C2_RIGHTTOLEFT)
            *pcs |= 1;
        if (typetab[index].charset == CHS_ARABIC)
            *pcs |= 2;
        if (typetab[index].charset == CHS_HEBREW)
            *pcs |= 4;
        out |= *pcs++;
    }
    return out;
}
```

UniQueryCntrl

UniQueryCntrl queries character attributes.

Format

```
#include <unidef.h>
```

int UniQueryCntrl
(const LocaleObject locale_object, UniChar uc)

Parameters

locale_object (const LocaleObject)
: A locale object created by UniCreateLocaleObject or NULL.

uc (UniChar)
: The UniChar character to query.

Returns

Return Value (int) - returns
: If the result of the test is true, the function returns 1. Otherwise, 0 is returned.

Remarks

This function provides the functionality of UniCreateAttrObject, UniQueryCharAttr, and UniFreeAttrObject as an atomic operation for the invariant attributes.

The locale may be specified as NULL to indicate default Unicode character attributes

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType
- UniQueryCharTypeTable

Related Functions

Example

```
This example shows how to query character attributes.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      uni_char = 0x000A;    /* Unicode newline character */
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Query character attribute */
         result = UniQueryCntrl(locale_object, uni_char);
         if (result)
           printf("UniChar character %04X is a control character\n", uni_char);
         else
           printf("UniChar character %04X is not a control character\n", uni_char);
return ULS_SUCCESS;
}
```

UniQueryCountryName

UniQueryCountryName returns the name of the country in the language specified.

Format

```
#include <unidef.h>
```

int UniQueryCountryName
  (UniChar * country, UniChar * isolang, UniChar * * infoitem)

Parameters

country (UniChar *)
  The two character ID of the country to query.

isolang (UniChar *)
  The two character ID of the language used to return the country name.

infoitem (UniChar * *)
  A pointer to the country name.

Returns

Return Value (int) - returns
  UniQueryCountryName returns one of the following values:
  ULS_INVALID
    The country ID supplied is not known.

  0 is returned upon success and the country name has been returned to the caller.

Remarks

  This function only queries system provided locales to determine valid country names.

Related Functions

- UniQueryLanguageName

Example

```
This example shows how to query a country name.
#include <stdio.h>
```

```
#include <unidef.h>
#include <ulsitem.h>

int main(void) {
LocaleObject locale_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      *pinfo;
UniChar      *langName;
UniChar      *countryName;
UniChar      *mriLanguage;
UniChar      uni_char = L'5';    /* Unicode number 5 character */

    /*******************************************************************/
    /* Assumes LANG environment variable set to a valid locale name, */
    /* such as fr_FR                                                 */
    /*******************************************************************/
    rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                     (UniChar *)L"", &locale_object);
    if (rc != ULS_SUCCESS) {
       printf("UniCreateLocaleObject error: return code = %u\n", rc);
       return 1;
    }

    /* Determine the language to get the country name in */
    rc = UniQueryLocaleItem(locale_object, LOCI_sLanguageID,
            &mriLanguage);

    if (rc != ULS_SUCCESS) {
        printf("UniQueryLocaleItem error: return code = %u\n", rc);
        return 1;
    }

    /* Get the ISO country ID
    rc = UniQueryLocaleItem(locale_object, LOCI_sCountryID, &pinfo);

    if (rc != ULS_SUCCESS) {
       printf("UniQueryLocaleItem error: return code = %u\n", rc);
       return 1;
    }

    /* Now we can determine the country name in the proper language */
    rc = UniQueryCountryName(pinfo, mriLanguage, &countryName);

    if (rc != ULS_SUCCESS) {
       printf("UniQueryCountryName error: return code = %u\n", rc);
       return 1;
    }

   printf("Country name is = %ls\n", countryName);
```

return ULS_SUCCESS;
}

UniQueryDigit

UniQueryDigit queries character attributes.

Format

```
#include <unidef.h>
```

int UniQueryDigit
(const LocaleObject locale_object, UniChar uc)

Parameters

locale_object (const LocaleObject)
A locale object created by UniCreateLocaleObject or NULL.

uc (UniChar)
The UniChar character to query.

Returns

Return Value (int) - returns
If the result of the test is true, the function returns 1. Otherwise, 0 is returned.

Remarks

This function provides the functionality of UniCreateAttrObject, UniQueryCharAttr, and UniFreeAttrObject as an atomic operation for the invariant attributes.

The locale may be specified as NULL to indicate default Unicode character attributes

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType
- UniQueryCharTypeTable

Example

```
This example shows how to query character attributes.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      uni_char = L'5';    /* Unicode number 5 character */
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Query character attribute */
         result = UniQueryDigit(locale_object, uni_char);
         if (result)
           printf("UniChar character %04X is a digit\n", uni_char);
         else
           printf("UniChar character %04X is not a digit\n", uni_char);
```

return ULS_SUCCESS;
}

UniQueryGraph

UniQueryGraph queries character attributes.

Format

```
#include <unidef.h>
```

int UniQueryGraph
(const LocaleObject locale_object, UniChar uc)

Parameters

locale_object (const LocaleObject)
A locale object created by UniCreateLocaleObject or NULL.

uc (UniChar)
The UniChar character to query.

Returns

Return Value (int) - returns
If the result of the test is true, the function returns 1. Otherwise, 0 is returned.

Remarks

This function provides the functionality of UniCreateAttrObject, UniQueryCharAttr, and UniFreeAttrObject as an atomic operation for the invariant attributes.

The locale may be specified as NULL to indicate default Unicode character attributes

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType

- UniQueryCharTypeTable

Example

```
This example shows how to query character attributes.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      uni_char = L'S';    /* Unicode Latin uppercase letter S */
         /*******************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                  */
         /*******************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Query character attribute */
         result = UniQueryGraph(locale_object, uni_char);
         if (result)
           printf("UniChar character %04X is a graphic character\n", uni_char);
         else
           printf("UniChar character %04X is not a graphic character\n", uni_char);
```

return ULS_SUCCESS;
}

UniQueryLanguageName

UniQueryLanguageName returns the name of the language in the language specified.

Format

```
#include <unidef.h>
```

int UniQueryLanguageName
(UniChar * language, UniChar * isolang, UniChar * * infoitem)

Parameters

language (UniChar *)
The two character ID of the language to query.

isolang (UniChar *)
The two character ID of the language used to return the language name.

infoitem (UniChar * *)

A pointer to the language name.

Returns

Return Value (int) - returns

UniQueryLanguageName returns one of the following values:

ULS_INVALID

The language ID supplied is not known.

0 is returned upon success and the language name has been returned to the caller.

Remarks

This function only queries system provided locales to determine valid language names.

Related Functions

- UniQueryCountryName

Example

```
This example shows how to query a language name.
#include <stdio.h>

#include <unidef.h>
#include <ulsitem.h>

int main(void) {
LocaleObject locale_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      *pinfo;
UniChar      *languageName;
UniChar      *mriLanguage;
UniChar      uni_char = L'5';    /* Unicode number 5 character */

    /*****************************************************************/
    /* Assumes LANG environment variable set to a valid locale name, */
    /* such as fr_FR                                                 */
    /*****************************************************************/
    rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                               (UniChar *)L"", &locale_object);
    if (rc != ULS_SUCCESS) {
       printf("UniCreateLocaleObject error: return code = %u\n", rc);
       return 1;
    }

    /* Determine the language to get the language name in */
    rc = UniQueryLocaleItem(locale_object, LOCI_sLanguageID,
            &mriLanguage);

    if (rc != ULS_SUCCESS) {
        printf("UniQueryLocaleItem error: return code = %u\n", rc);
        return 1;
    }
```

```
     /* Get the ISO country ID
     rc = UniQueryLocaleItem(locale_object, LOCI_sLanguageID, &pinfo);

     if (rc != ULS_SUCCESS) {
        printf("UniQueryLocaleItem error: return code = %u\n", rc);
        return 1;
     }

     /* Now we can determine the country name in the proper language */
     rc = UniQueryCountryName(pinfo, mriLanguage, &languageName);

     if (rc != ULS_SUCCESS) {
        printf("UniQueryCountryName error: return code = %u\n", rc);
        return 1;
     }

    printf("Language name is = %ls\n", languageName);
```

return ULS_SUCCESS;
}

UniQueryLocaleInfo

UniQueryLocaleInfo retrieves information about locale conventions.

Format

```
#include <unidef.h>
```

int UniQueryLocaleInfo
(const LocaleObject locale_object, struct UniLconv **UniLconv_addr_ptr)

Parameters

locale_object (const LocaleObject)
A locale object created by UniCreateLocaleObject.

UniLconv_addr_ptr (struct UniLconv **)
The address of a pointer to receive a structure filled with locale conventions.

Returns

return value (int) - returns
UniQueryLocaleInfo returns one of the following values:
ULS_SUCCESS
The UniLconv structure was successfully filled with items associated with the locale object locale_object.
ULS_BADOBJ
The locale object specified by locale_object is not a valid locale object.

Remarks

UniQueryLocaleInfo retrieves information from the locale indicated by the locale_object argument and places the information in a UniLconv structure. UniQueryLocaleInfo allocates memory to hold the UniLconv structure. It is the application's responsibility to free the memory with UniFreeLocaleInfo when the UniLconv structure is no longer needed. The address of the UniLconv structure is returned in UniLconv_struct. The UniLconv structure is filled in, according to the locale indicated by the locale object handle argument.

The UniLconv structure contains the following members:

UniChar *decimal_point;
: /* non-monetary decimal point */

UniChar *thousands_sep;
: /* non-monetary thousands separator */

UniChar *grouping;
: /* non-monetary size of grouping */

UniChar *int_curr_symbol;
: /* international currency symbol and separator */

UniChar *currency_symbol;
: /* local currency symbol */

UniChar *mon_decimal_point;
: /* monetary decimal point */

UniChar *mon_thousands_sep;
: /* monetary thousands separator */

UniChar *mon_grouping;
: /* monetary size of grouping */

UniChar *positive_sign;
: /* non-negative values sign */

UniChar *negative_sign;
: /* negative values sign */

UniChar int_frac_digits;
: /* number of fractional digits - int currency */

UniChar frac_digits;
: /* number of fractional digits - local currency */

UniChar p_cs_precedes;
: /* (non-neg curr sym) 1-precedes, 0-succeeds */

UniChar p_sep_by_space;
: /* (non-neg curr sym) 1-space, 0-no space */

UniChar n_cs_precedes;
: /* (neg curr sym) 1-precedes, 0-succeeds */

UniChar n_sep_by_space;
: /* (neg curr sym) 1-space, 0-no space */

UniChar p_sign_posn;
: /* positioning of non-negative monetary sign */

UniChar n_sign_posn;
: /* positioning of negative monetary sign */

short os2_mondecpt;
: /* os2 curr sym positioning */

UniChar *debit_sign;
: /* non-negative-valued monetary symbol - "DB"*/

UniChar *credit_sign;
/* negative-valued monetary symbol - "CR" */
UniChar *left_parenthesis;
/* negative-valued monetary symbol - "(" */
UniChar *right_parenthesis;
/* negative-valued monetary symbol - ")" */

The value of grouping and mon_grouping is interpreted according to the following:

0xffff
No further grouping is to be performed.
0x0000
The previous element is to be repeatedly used for the remainder of the digits.
other
The integer value is the number of digits that comprise the current group.

The next element is examined to determine the size of the next group of digits before the current group.

The n_sign_posn and p_sign_posn elements are interpreted according to the following:

0
Quantity and currency_symbol are enclosed in parentheses
1
Sign precedes the quantity and currency_symbol
2
Sign follows the quantity and currency_symbol
3
Sign precedes the currency_symbol
4
Sign follows the currency_symbol
5
Use debit or credit sign for p_sign_posn or n_sign_posn

Related Functions

- UniFreeLocaleInfo

Example

```
This example shows how to retrieve information about locale conventions.
#include <stdio.h>
#include <unidef.h>

int main(void) {
LocaleObject      locale_object = NULL;
struct UniLconv   *puni_lconv = NULL;

int               rc = ULS_SUCCESS;
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
```

```
        /* such as fr_FR                                                   */
        /*******************************************************************/
        rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                   (UniChar *)L"", &locale_object);
        if (rc != ULS_SUCCESS) {
          printf("UniCreateLocaleObject error: return code = %u\n", rc);
          return 1;
        }
        /* Retrieve locale information */
        rc = UniQueryLocaleInfo(locale_object, &puni_lconv);
        if (rc != ULS_SUCCESS) {
          printf("UniQueryLocaleInfo error: return code = %u\n", rc);
          return 1;
        }
        /* print the value of the os2 currency symbol position */
        printf("The os2 currency symbol position is %d\n",
                puni_lconv->os2_mondecpt);
        return ULS_SUCCESS;

}
```

UniQueryLocaleItem

UniQueryLocaleItem retrieves locale information by item.

Format

```
#include <unidef.h>
```

int UniQueryLocaleItem
(const LocaleObject locale_object, LocaleItem item, UniChar **info_item_addr_ptr)

Parameters

locale_object (const LocaleObject)
A locale object created by UniCreateLocaleObject.

item (LocaleItem)
The item to be queried.

info_item_addr_ptr (UniChar **)
Address of a pointer where the locale information will be received.

Returns

return value (int) - returns
UniQueryLocaleItem returns one of the following values:
ULS_SUCCESS
The info_item_addr_ptr string is successfully filled with item associated with the locale object locale_object.
ULS_INVALID

The locale item is not a valid locale item.

Remarks

UniQueryLocaleItem returns a pointer in info_item_addr_ptr to a null-terminated UniChar string containing information found in the locale object identified by locale_object about the language or cultural item named by the item argument. UniQueryLocaleItem allocates the memory to hold the UniChar string and returns a pointer in info_item_addr_ptr. Use UniFreeMem to free the memory associated with info_item_addr_ptr by UniQueryLocaleItem.

The constant names and values for item are contained in ulsitem.h:

Item Name	**Item Description**
LOCI_sDateTime	Date and time format string
LOCI_sShortDate	Short date format
LOCI_sTimeFormat	Time format string
LOCI_s1159	AM string
LOCI_s2359	PM sring
LOCI_sAbbrevDayName7	Abbreviation of day 7 (Sun)
LOCI_sAbbrevDayName1	Abbreviation of day 1 (Mon)
LOCI_sAbbrevDayName2	Abbreviation of day 2 (Tue)
LOCI_sAbbrevDayName3	Abbreviation of day 3 (Wed)
LOCI_sAbbrevDayName4	Abbreviation of day 4 (Thu)
LOCI_sAbbrevDayName5	Abbreviation of day 5 (Fri)
LOCI_sAbbrevDayName6	Abbreviation of day 6 (Sat)
LOCI_sDayName7	Name of day of week 7 (Sun)
LOCI_sDayName1	Name of day of week 1 (Mon)
LOCI_sDayName2	Name of day of week 2 (Tue)
LOCI_sDayName3	Name of day of week 3 (Wed)
LOCI_sDayName4	Name of day of week 4 (Thu)
LOCI_sDayName5	Name of day of week 5 (Fri)

LOCI_sDayName6	Name of day of week 6 (Sat)
LOCI_sAbbrevMonthName1	Abbreviation of month 1
LOCI_sAbbrevMonthName2	Abbreviation of month 2
LOCI_sAbbrevMonthName3	Abbreviation of month 3
LOCI_sAbbrevMonthName4	Abbreviation of month 4
LOCI_sAbbrevMonthName5	Abbreviation of month 5
LOCI_sAbbrevMonthName6	Abbreviation of month 6
LOCI_sAbbrevMonthName7	Abbreviation of month 7
LOCI_sAbbrevMonthName8	Abbreviation of month 8
LOCI_sAbbrevMonthName9	Abbreviation of month 9
LOCI_sAbbrevMonthName10	Abbreviation of month 10
LOCI_sAbbrevMonthName11	Abbreviation of month 11
LOCI_sAbbrevMonthName12	Abbreviation of month 12
LOCI_sMonthName1	Name of month 1
LOCI_sMonthName2	Name of month 2
LOCI_sMonthName3	Name of month 3
LOCI_sMonthName4	Name of month 4
LOCI_sMonthName5	Name of month 5
LOCI_sMonthName6	Name of month 6
LOCI_sMonthName7	Name of month 7
LOCI_sMonthName8	Name of month 8
LOCI_sMonthName9	Name of month 9
LOCI_sMonthName10	Name of month 10
LOCI_sMonthName11	Name of month 11
LOCI_sMonthName12	Name of month 12

LOCI_sDecimal	Decimal point
LOCI_sThousand	Triad separator
LOCI_sYesString	Yes string
LOCI_sNoString	No string
LOCI_sCurrency	Currency symbol
LOCI_sCodeSet	Locale codeset
LOCI_xLocaleToken	IBM Locale Token
LOCI_xWinLocale	Win32 Locale ID
LOCI_iLocaleResnum	Resource number for description
LOCI_sNativeDigits	String of native digits
LOCI_iMaxItem	Maximum item number
LOCI_sTimeMark	Time mark (am/pm) format
LOCI_sEra	Era definition
LOCI_sAltShortDate	Alternate short date format string
LOCI_sAltDateTime	Alternate date and time format
LOCI_sAltTimeFormat	Alternate time format
LOCI_sAltDigits	XPG4 alternate digist
LOCI_sYesExpr	xpg4 yes expression
LOCI_sNoExpr	xpg4 no expression
LOCI_sDate	Short date separator
LOCI_sTime	Time separator
LOCI_sList	List separator
LOCI_sMonDecimalSep	Monetary currency separator
LOCI_sMonThousandSep	Monetary triad separator
LOCI_sGrouping	Grouping of digits

LOCI_sMonGrouping	Monetary groupings
LOCI_iMeasure	Measurement (Metric, British)
LOCI_iPaper	Normal paper size
LOCI_iDigits	Digits to right of decimal
LOCI_iTime	Clock format
LOCI_iDate	Format of short date
LOCI_iCurrency	Format of currency
LOCI_iCurrDigits	Digits to right for currency
LOCI_iLzero	Leading zero used
LOCI_iNegNumber	Format of negative number
LOCI_iLDate	Format of long date
LOCI_iCalendarType	Type of default calandar
LOCI_iFirstDayOfWeek	First day of week (0=Mon)
LOCI_iFirstWeekOfYear	First week of year
LOCI_iNegCurr	Format of negative currency
LOCI_iTLzero	Leading zero on time
LOCI_iTimePrefix	AM/PM preceeds time
LOCI_iOptionalCalendar	Alternate calandar type
LOCI_sIntlSymbol	International currency symbol
LOCI_sAbbrevLangName	Windows language abbreviation
LOCI_sCollate	Collation table
LOCI_iUpperType	Upper case algorithm
LOCI_iUpperMissing	Action for missing upper case
LOCI_sPositiveSign	Positive sign
LOCI_sNegativeSign	Negative sign

LOCI_sLeftNegative	Left paren for negative
LOCI_sRightNegative	Right paren for negative
LOCI_sLongDate	Long date formatting string
LOCI_sAltLongDate	Alternate long date format string
LOCI_sMonthName13	Name of month 13
LOCI_sAbbrevMonthName13	Abbreviation of month 13
LOCI_sName	OS/2 locale name
LOCI_sLanguageID	Abbreviation for language (ISO)
LOCI_sCountryID	Abbreviation for country (ISO)
LOCI_sEngLanguage	English name of Language
LOCI_sLanguage	Native name of language
LOCI_sEngCountry	English name of country
LOCI_sCountry	Localized country name
LOCI_sNativeCtryName	Name of country in native language
LOCI_iCountry	Country code
LOCI_sISOCodepage	ISO codepage name
LOCI_iAnsiCodepage	Windows codepage
LOCI_iCodepage	OS/2 primary codepage
LOCI_iAltCodepage	OS/2 alternate codepage
LOCI_iMacCodepage	Mac codepage
LOCI_iEbcdicCodepage	Ebcdic codepage
LOCI_sOtherCodepages	Other ASCII codepages
LOCI_sSetCodepage	Codpage to set on activation
LOCI_sKeyboard	Primary keyboard name
LOCI_sAltKeyboard	Alternate keyboard name

LOCI_sSetKeyboard	Keyboard to set on activation
LOCI_sDebit	Debit string
LOCI_sCredit	Credit string
LOCI_sLatin1Locale	Locale for Latin 1 names
LOCI_wTimeFormat	Win32 Time format
LOCI_wShortDate	Win32 Date format
LOCI_wLongDate	Win32 Long date format

Related Functions

- UniSetUserLocaleItem

Example

```
This example shows how to retrieve locale information by item.
#include <stdio.h>
#include <unidef.h>

int main(void) {
LocaleObject      locale_object = NULL;
UniChar           *pinfo_item;
int               rc = ULS_SUCCESS;
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Retrieve name of the tenth month locale item */
         rc = UniQueryLocaleItem(locale_object,
                                 MON_10,
                                 &pinfo_item);
         if (rc != ULS_SUCCESS) {
           printf("UniQueryLocaleItem error: return code = %u\n", rc);
           return 1;
         }
         rc = UniFreeMem(pinfo_item);
         if (rc != ULS_SUCCESS) {
           printf("UniFreeMem error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;

}
```

UniQueryLocaleList

UniQueryLocaleList returns a buffer filled with a list of the locales defined on the system.

Format

```
#include <unidef.h>
```

int UniQueryLocaleList

(int flag, UniChar * uniBuffer, int numUniChars)

Parameters

flag (int)
: A flag indicating whether to return a list of system defined locales or user defined locales.

uniBuffer (UniChar *)
: A pointer to a buffer that is filled with UniChar's representing the list of system or user defined locales.

numUniChars (int)
: The maximum size of the buffer used to return the list of locales.

Returns

return value (int) - returns
: UniQueryLocaleList returns one of the following values:
: **ULS_SUCCESS**
: The buffer is filled with the appropriate list of locales.
: **ULS_BUFFERFULL**
: The locale list size exceeded the supplied buffer size.

Remarks

The flag parameter can be used to select either system defined locales or user defined locales. The system and user defined choices can be or'ed together to retrieve the complete list of locales.

Related Functions

- UniQueryLocaleValue

Example

```
This example shows how to retrieve a list of locales available on the
system.
#include <stdio.h>

#include <unidef.h>
```

```
int main(void) {
int     rc = ULS_SUCCESS;

/* Arrays containing system and user locales */
UniChar *uniSysLocales;
UniChar *uniUsrLocales;

    /* Allocate space for the locale list (2 bytes/UniChar) */
    uniSysLocales = (UniChar *) malloc(4096);
    uniUsrLocales = (UniChar *) malloc(4096);

    if(!uniSysLocales || !uniUsrLocales) {
      printf("Malloc failed error: return code = %u\n", rc);
      return 1;
    }

    /*********************************************************/
    /* Obtain the list of system and user defined locales */
    /*********************************************************/
    rc = UniQueryLocaleList(UNI_SYSTEM_LOCALES, uniSysLocales, 2048);
    if(rc) {
      printf("UniQueryLocaleList error: return code = %u\n", rc);
      return 1;
    }


    rc = UniQueryLocaleList(UNI_USER_LOCALES, uniUsrLocales, 2048);

    if(rc) {
      printf("UniQueryLocaleList error: return code = %u\n", rc);
      return 1;
    }

    return ULS_SUCCESS;

}
```

UniQueryLocaleObject

UniQueryLocaleObject retrieves the locale name.

Format

```
#include <unidef.h>
```

int UniQueryLocaleObject
(const LocaleObject locale_object, int category, int LocaleSpecType, void **locale_name)

Parameters

locale_object (const LocaleObject)
Locale object created by a call to UniCreateLocaleObject.

category (int)

Locale category identifier.

The permissible values for **category** are:

- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_MONETARY**
- **LC_NUMERIC**
- **LC_TIME**

LocaleSpecType (int)
The **LocaleSpecType** argument can take any of the following values, which are constants defined in the header **unidef.h**:
UNI_TOKEN_POINTER
Requests that a pointer to a token pointer be returned.
UNI_MBS_STRING_POINTER
Requests that a multibyte string pointer be returned.
UNI_UCS_STRING_POINTER
Requests that a UCS string pointer be returned.

locale_name (void **)
The address of a pointer variable locale_name that will contain the locale name.

Returns

return value (int) - returns
UniQueryLocaleObject returns one of the following values:
ULS_SUCCESS
A valid locale specification for the supplied locale object is returned.
ULS_INVALID
The locale specified by locale_object is invalid.
ULS_OTHER
The C locale is because LOCALE.DLL cound not be found.
ULS_UNSUPPORTED
The locale name is valid but the locale cound not be found.

Remarks

UniQueryLocaleObject returns a pointer to a locale specification in the area pointed to by **locale_name**. UniQueryLocaleObject allocates memory to hold the generated value as necessary. Use UniFreeMem to free the memory associated with locale_name by UniQueryLocaleObject.

The value returned in the area pointed to by **locale_name** will point to either a string or a token, as indicated by the value of the **LocaleSpecType** argument.

When the **LocaleSpecType** argument is **UNI_TOKEN_POINTER** and the category argument is valid, a pointer to a token that represents the locale value associated with the **category** argument is returned, if such a token exists.

When the **LocaleSpecType** argument is **UNI_MBS_STRING_POINTER** or **UNI_UCS_STRING_POINTER**, UniQueryLocaleObject returns a pointer to a string that represents the locale value associated with the **category** argument.

When the **LocaleSpecType** argument is **UNI_MBS_STRING_POINTER** or **UNI_UCS_STRING_POINTER** and the **category** argument is **LC_ALL**, a string that represents the values of all of the locale categories of **locale_object** is returned. The returned string may be used as the **LocaleSpec** argument to UniCreateLocaleObject to create a locale object that is a functional equivalent of **locale_object**.

When the **LocaleSpecType** argument is **UNI_MBS_STRING_POINTER** or **UNI_UCS_STRING_POINTER** and the **category** argument is **LC_COLLATE**, **LC_CTYPE**, **LC_MESSAGES**, **LC_MONETARY**, **LC_NUMERIC**, or **LC_TIME**, a string that represents the value of the respective locale category of **locale_object** is returned. The returned string may be used as the **LocaleSpec** argument to UniCreateLocaleObject create a locale object. All locale category values are set to the value of the queried locale category of **locale_object**.

If **locale_object** contains a NULL pointer, UniQueryLocaleObject returns a locale specification pointer identifying the respective categories of the default locale. If the **category** argument is **LC_ALL**, this value can be passed to UniCreateLocaleObject to create a locale object that is the functional equivalent of the current default locale, as specified by the environment variables of the current process.

If **locale_object** is invalid, the contents of **locale_name** are undefined and no memory is allocated.

Related Functions

- UniCreateLocaleObject
- UniFreeLocaleObject

Example

```
This example shows how to retrieve a locale category name.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject      locale_object = NULL;
UniChar           *plocale_name;
int               rc = ULS_SUCCESS;
```

/* Retrieve locale name of default locale */

```
         rc = UniQueryLocaleObject(NULL,
                                   LC_ALL,
                                   UNI_UCS_STRING_POINTER,
                                   (void **)&plocale_name);
         if (rc != ULS_SUCCESS) {
           printf("UniQueryLocaleObject error: return code = %u\n", rc);
           return 1;
         }
```

```
          /* Create a locale object based upon the default setting */
          rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                     (UniChar *)plocale_name, &locale_object);
          if (rc != ULS_SUCCESS) {
            printf("UniCreateLocaleObject error: return code = %u\n", rc);
            return 1;
          }
          rc = UniFreeMem(plocale_name);
          if (rc != ULS_SUCCESS) {
            printf("UniFreeMem error: return code = %u\n", rc);
            return 1;
          }
          return ULS_SUCCESS;

}
```

UniQueryLocaleValue

UniQueryLocaleValue returns an integral value associated with the requested locale item.

Format

```
#include <unidef.h>
```

int UniQueryLocaleValue

(const LocaleObject locale_object, LocaleItem item, int * info_item)

Parameters

locale_object (const LocaleObject)
: Locale object created by a call to UniCreateLocaleObject.

item (LocaleItem)
: The locale item being requested.

info_item (int *)
: A pointer to an integer where the value of the locale item is returned.

Returns

return value (int) - returns
: UniQueryLocaleValue returns one of the following values:
: **ULS_SUCCESS**
    The info_item is set to the appropriate integral item value.
: **ULS_BADOBJ**
    The locale item is invalid or does not exist.

Remarks

When a locale item is requested that does not have an integral value, zero is returned to the caller.

Related Functions

- UniQueryLocaleList

Example

```
This example shows how to retrieve the value for a locale item.
#include <stdio.h>

#include <unidef.h>
#include <ulsitem.h>
int main(void) {
LocaleObject locale_object = NULL;
ULONG        pmCodepage;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      uni_char = L'd';    /* Unicode lowercase Latin letter d */
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }

         rc = UniQueryLocaleValue(locale_object, LOCI_iCodepage,
                  (int *)&pmCodepage);

         if (rc != ULS_SUCCESS) {
           printf("UniQueryLocaleValue error: return code = %u\n", rc);
           return 1;
         }

         printf("Presentation manager is using codepage %d\n",
                 pmCodepage);

    return ULS_SUCCESS;

}
```

UniQueryLower

UniQueryLower queries character attributes.

Format

```
#include <unidef.h>
```

int UniQueryLower

(const LocaleObject locale_object, UniChar uc)

Parameters

locale_object (const LocaleObject)
    A locale object created by UniCreateLocaleObject or NULL.
uc (UniChar)
    The UniChar character to query.

Returns

Return Value (int) - returns
    If the result of the test is true, the function returns 1. Otherwise, 0 is returned.

Remarks

This function provides the functionality of UniCreateAttrObject UniQueryCharAttr, and UniFreeAttrObject as an atomic operation for the invariant attributes.

The locale may be specified as NULL to indicate default Unicode character attributes

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType
- UniQueryCharTypeTable

Example

```
This example shows how to query character attributes.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      uni_char = L'd';    /* Unicode lowercase Latin letter d */
         /*******************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                  */
         /*******************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Query character attribute */
         result = UniQueryLower(locale_object, uni_char);
         if (result)
           printf("UniChar character %04X is a lowercase character\n", uni_char);
         else
           printf("UniChar character %04X is not a lowercase character\n", uni_char);
```

return ULS_SUCCESS;
}

UniQueryNumericValue

UniQueryNumericValue returns the numeric value associated with a Unicode character.

Format

```
#include <unidef.h>
```

int UniQueryNumericValue
(UniChar uc)

Parameters

uc (UniChar)
The UniChar character to query.

Returns

Return Value (int) - returns
The function returns a -1 if unsuccessful. Otherwise, the numeric value of the Unicode character is returned.

Remarks

This function returns numeric values for ranges of Unicode numeric characters. The function can be used to identify the digits both decimal and hexadecimal for Latin numbers, Arabic, Indian dialects, Laotian, Thai, Han and others represented in the Unicode character set.

Related Functions

- UniQueryCharType

Example

```
This example shows how to query the numeric value of a character.
#include <stdio.h>

#include <unidef.h>
int main(void) {

int          result = 0;
int          rc = ULS_SUCCESS;
UNICTYPE   * ct;
UniChar    * uptr;
```

```
         /* Set up a Unicode numeric character to test */
         uptr = L'1';

         /* Determine the characters type */
         ct = UniQueryCharType(*uptr);

         /* Test the Unicode character to see if it is a digit */
         /* It can be either decimal or Hex */
         if ((ct->itype & CT_XDIGIT) || (ct->itype & CT_NUMBER)) {
         num = UniQueryNumericValue(*uptr);

         if (num == -1) {
           printf("UniQueryNumericValue error: return code = %u\n", rc);
           return 1;
         }
        return ULS_SUCCESS;

}
```

UniQueryPrint

UniQueryPrint queries character attributes.

Format .

```
#include <unidef.h>
```

int UniQueryPrint
(const LocaleObject locale_object, UniChar uc)

Parameters

locale_object (const LocaleObject)
A locale object created by UniCreateLocaleObject or NULL.

uc (UniChar)
The UniChar character to query.

Returns

Return Value (int) - returns
If the result of the test is true, the function returns 1. Otherwise, 0 is returned.

Remarks

This function provides the functionality of UniCreateAttrObject, UniQueryCharAttr, and UniFreeAttrObject as an atomic operation for the invariant attributes.

The locale may be specified as NULL to indicate default Unicode character attributes

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType
- UniQueryCharTypeTable

Example

```
This example shows how to query character attributes.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      uni_char = L'd';    /* Unicode lowercase Latin letter d */
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Query character attribute */
         result = UniQueryPrint(locale_object, uni_char);
         if (result)
           printf("UniChar character %04X is a printable character\n", uni_char);
         else
           printf("UniChar character %04X is not a printable character\n", uni_char);
```

return ULS_SUCCESS;
}

UniQueryPunct

UniQueryPunct queries character attributes.

Format

```
#include <unidef.h>
```

int UniQueryPunct
(const LocaleObject locale_object, UniChar uc)

Parameters

locale_object (const LocaleObject)
A locale object created by UniCreateLocaleObject or NULL.

uc (UniChar)
    The UniChar character to query.

Returns

Return Value (int) - returns
    If the result of the test is true, the function returns 1. Otherwise, 0 is returned.

Remarks

This function provides the functionality of UniCreateAttrObject, UniQueryCharAttr, and UniFreeAttrObject as an atomic operation for the invariant attributes.

The locale may be specified as NULL to indicate default Unicode character attributes

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType
- UniQueryCharTypeTable

Example

```
This example shows how to query character attributes.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      uni_char = L'?';    /* Unicode Latin question mark */
         /*********************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                           */
         /*********************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Query character attribute */
         result = UniQueryPrint(locale_object, uni_char);
         if (result)
           printf("UniChar character %04X is a punctuation character\n", uni_char);
         else
           printf("UniChar character %04X is not a punctuation character\n", uni_char);
```

return ULS_SUCCESS;
}

UniQuerySpace

UniQuerySpace queries character attributes.

Format

```
#include <unidef.h>
```

int UniQuerySpace
    (const LocaleObject locale_object, UniChar uc)

Parameters

locale_object (const LocaleObject)
    A locale object created by UniCreateLocaleObject or NULL.

uc (UniChar)
    The UniChar character to query.

Returns

Return Value (int) - returns
    If the result of the test is true, the function returns 1. Otherwise, 0 is returned.

Remarks

This function provides the functionality of UniCreateAttrObject, UniQueryCharAttr, and UniFreeAttrObject as an atomic operation for the invariant attributes.

The locale may be specified as NULL to indicate default Unicode character attributes

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType
- UniQueryCharTypeTable

Example

```
This example shows how to query character attributes.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      uni_char = L' ';    /* Unicode space character */
```

```
        /******************************************************************/
        /* Assumes LANG environment variable set to a valid locale name, */
        /* such as fr_FR                                                  */
        /******************************************************************/
        rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                   (UniChar *)L"", &locale_object);
        if (rc != ULS_SUCCESS) {
          printf("UniCreateLocaleObject error: return code = %u\n", rc);
          return 1;
        }
        /* Query character attribute */
        result = UniQuerySpace(locale_object, uni_char);
        if (result)
          printf("UniChar character %04X is a space character\n", uni_char);
        else
          printf("UniChar character %04X is not a space character\n", uni_char);
```

return ULS_SUCCESS;
}

UniQueryStringType

UniQueryStringType is used to query character types for a string.

Format

```
#include <unidef.h>
```

ULONG UniQueryStringType
(UniChar * ustr, int size, USHORT * outstr, int kind)

Parameters

ustr (UniChar *)
A pointer to the Unicode string.

size (int)
The size of the UniChar character string to query.

outstr (USHORT *)
A pointer to an array of USHORTs. Each USHORT represents the type of one of the Unicode characters.

kind (int)
An integer describing the type of string.

Returns

Return Value (ulong) - UniQueryStringType returns one of the following:
ULS_SUCCESS

The function was successful.

ULS_INVALID

The kind of string supplied is not known.

Remarks

`Valid values for kind are:`

- CT_ITYPE
- CT_BIDI
- CT_CHARSET
- CT_EXTENDED
- CT_CODEPAGE
- CT_INDEX
- CT_CTYPE1 - Win32 compatible XPG/4
- CT_CTYPE2 - Win32 compatible BiDi
- CT_CTYPE3 - Win32 compatible extended

Related Functions

- UniQueryCharType
- UniQueryCharTypeTable

Example

```
This example shows how to query string types.
#include <unidef.h>
#include <stdlib.h>

/*
* CountJapanese: Count the number of Japanese chars in string
*/
int CountJapanese(UniChar * instr) {
int len;
USHORT * outbuf, * pout;
int count;

    /*
     * Get some memory to return the string
     */
    len = UniStrlen(instr);
    outbuf = malloc(len * sizeof(UniChar));

    /*
     * Query the extended character type of the string
     */
    UniQueryStringType(instr, len, outbuf, CT_EXTENDED);

    /*
     * Search the retuned array of types for Japanese chars
     */
    count = 0;
    pout = outbuf;
    while (len--) {
        if (*pout & (C3_KATAKANA|C3_HIRAGANA|C3_IDEOGRAPH)) {
            count++;
        }
        pout++;
    }

    /*
     * Free up type array and return
     */
    free(outbuf);
    return count;
}
```

UniQueryUpper

UniQueryUpper queries character attributes.

Format

```
#include <unidef.h>
```

int UniQueryUpper
(const LocaleObject locale_object, UniChar uc)

Parameters

locale_object (const LocaleObject)
A locale object created by UniCreateLocaleObject or NULL.

uc (UniChar)
The UniChar character to query.

Returns

Return Value (int) - returns
If the result of the test is true, the function returns 1. Otherwise, 0 is returned.

Remarks

This function provides the functionality of UniCreateAttrObject, UniQueryCharAttr, and UniFreeAttrObject as an atomic operation for the invariant attributes.

The locale may be specified as NULL to indicate default Unicode character attributes

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType
- UniQueryCharTypeTable

Example

```
This example shows how to query character attributes.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      uni_char = L'D';    /* Unicode uppercase Latin letter D */
         /*******************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                   */
         /*******************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Query character attribute */
         result = UniQueryUpper(locale_object, uni_char);
         if (result)
           printf("UniChar character %04X is an uppercase character\n", uni_char);
         else
           printf("UniChar character %04X is not an uppercase character\n", uni_char);
```

return ULS_SUCCESS;
}

UniQueryXdigit

UniQueryXdigit queries character attributes.

Format

```
#include <unidef.h>
```

int UniQueryXdigit
(const LocaleObject locale_object, UniChar uc)

Parameters

locale_object (const LocaleObject)
A locale object created by UniCreateLocaleObject or NULL.

uc (UniChar)
The UniChar character to query.

Returns

Return Value (int) - returns
    If the result of the test is true, the function returns 1. Otherwise, 0 is returned.

Remarks

This function provides the functionality of UniCreateAttrObject, UniQueryCharAttr, and UniFreeAttrObject as an atomic operation for the invariant attributes.

The locale may be specified as NULL to indicate default Unicode character attributes

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType
- UniQueryCharTypeTable

Example

```
This example shows how to query character attributes.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
UniChar      uni_char = L'D';    /* Unicode uppercase Latin letter D */
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                      (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Query character attribute */
         result = UniQueryXdigit(locale_object, uni_char);
         if (result)
           printf("UniChar character %04X is a hex digit\n", uni_char);
         else
           printf("UniChar character %04X is not a hex digit\n", uni_char);
```

return ULS_SUCCESS;
}

UniScanForAttr

UniScanForAttr scans a Unicode string for an attribute match.

Format

```
#include <unidef.h>
```

int UniScanForAttr
(AttrObject attr_object, const UniChar *ucs, size_t num_elems, ulsBool inverse_op, size_t *offset)

Parameters

attr_object (AttrObject)
An attribute object created by UniCreateAttrObject.
ucs (const UniChar *)
The array of UniChar code elements to be scanned for the set of attributes in attr_object.
num_elems (size_t)
The number of UniChar code elements to be searched.
inverse_op (ulsBool)
Determines scanning rules. Set to false to search for the first match. Set to true to search for the first nonmatching element.
offset (size_t *)
An integer identifying the location of the first element meeting the criteria.

Returns

return value (int) - returns

UniScanForAttr returns one of the following:

ULS_SUCCESS
The function was successful.
ULS_NOMATCH
No code element meets the specified criteria.
ULS_BADOBJ
The attribute object specified by attr_object is not a valid attribute object.

Remarks

UniScanForAttr scans the array of code elements identified by ucs, from the position specified by ucs, searching for the first code element that matches or does not match the set of attributes specified by attr_object.

The inverse_op argument determines the rules for scanning and is an integer type containing one of the following values:

0 - FALSE

1 - TRUE

If inverse_op is set to FALSE, the function searches for the first code element that matches all of the attributes of the specified attr_object; if inverse_op is set to TRUE, the function searches for the first code element that matches none of the attributes of attr_object.

The search begins from the code element identified by ucs, through the next num_elems code elements. A non-negative integer identifying the location of the first code element meeting all of the criteria specified by attr_object is returned in the area pointed to by offset. This indicates the number of code elements offset from the code element identified by ucs, to the code element at which the attribute match is satisfied. If no code element meets the specified criteria, the contents of offset are undefined.

Related Functions

- UniQueryAttr
- UniQueryChar
- UniQueryCharAttr
- UniQueryCharType
- UniQueryCharTypeTable

Example

```
This example shows how to scan a Unicode string for an attribute match.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;

AttrObject   attr_object = NULL;
int          result = 0;
int          rc = ULS_SUCCESS;
size_t       offset = 0;
UniChar      *uni_char = L"os2";
         /*******************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                  */
         /*******************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* Create an attribute object */
         rc = UniCreateAttrObject(locale_object,
                                  (UniChar *)L"digit", &attr_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateAttrObject error: return code = %u\n", rc);
           return 1;
         }
         /* Make call to determine if string matches attributes */
         rc = UniScanForAttr(attr_object, uni_char, UniStrlen(uni_char),
                             FALSE, &offset);
         if (rc != ULS_SUCCESS) {
           printf("UniScanForAttr error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;

}
```

UniSetUserLocaleItem

UniSetUserLocaleItem is used to set a locale override for a locale item.

Format

```
#include <unidef.h>
```

int UniSetUserLocaleItem
(UniChar * locale, int item, int type, void * value)

Parameters

locale (UniChar *)
Locale to have item changed.

item (int)
Item to be set.

type (int)
Type of item.

value (void *)
Pointer to item to be changed.

Returns

return value (int) - returns

UniSetUserLocaleItem returns one of the following:

ULS_SUCCESS
The function was successful.
ULS_INVALID
No code element meets the specified criteria.
ULS_NOMATCH
The specified locale was not found.
ULS_NOMEMORY
A memory allocation error was detected.
ULS_BADOBJECT
The specified locale is not a user defined locale.
ULS_BUFFERFULL
The item is too long for the buffer.

Remarks

UniSetUserLocaleItem modifies an item in a user locale. This affects all users of that locale. After doing this function, a call to UniCompleteUserLocale is necessary to make the changes permanent.

Related Functions

- UniQueryLocaleItem
- UniCompleteUserLocale

Example

```
This example shows how to set a user locale item.
#include <stdio.h>

#include <unidef.h>
int main(void) {

UniChar      unilocaleName[John's_Locale];
LocaleObject locale_object = NULL;


int          result = 0;
int          rc = ULS_SUCCESS;
size_t       offset = 0;

         /******************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                  */
         /******************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }

         /* Set the time separator for John's locale to a semicolon */
         rc = UniSetUserLocaleItem(unilocaleName, TIMESEP,
                    ULTYPE_UNICODE, (UniChar *)L":");

         if (rc != ULS_SUCCESS) {
           printf("UniSetUserLocaleItem error: return code = %u\n", rc);
           return 1;
         }

         return ULS_SUCCESS;
}
```

UniStrcat

UniStrcat concatenates code element strings.

Format

```
#include <unidef.h>
```

UniChar *UniStrcat
(UniChar *ucs1, const UniChar *ucs2)

Parameters

ucs1 (UniChar *)
String to be appended to.

ucs2 (const UniChar *)
String to concatenate.

Returns

return value (UniChar *) - returns
Concatenated string.

Remarks

UniStrcat appends a copy of the code element string pointed to by ucs2 (including the terminating null code element) to the end of the code element string pointed to by ucs1. The initial wide code element of ucs2 overwrites the null code element at the end of ucs1. If copying takes place between objects that overlap, the results are unpredictable.

Related Functions

- UniStrncat

Example

```
This example shows how to concatenate Unicode strings.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    ucs1[40] = L"computer";

UniChar    *puni;
         puni = UniStrcat(ucs1, (UniChar *)L" program");
         return (ULS_SUCCESS);
}
```

UniStrchr

UniStrchr searches for the first occurrence of a code element.

Format

```
#include <unidef.h>
```

UniChar *UniStrchr
    (const UniChar *ucs, UniChar uc)

Parameters

ucs (const UniChar *)
    A null-terminated UniChar string to be searched.

uc (UniChar)
    The UniChar search character.

Returns

return value (UniChar *) - returns
    UniStrchr returns either a pointer to the located code element or a null pointer, if the code element does not occur in the string.

Remarks

UniStrchr locates the first occurrence of uc in the array of code elements pointed to by ucs. The terminating null code element is considered to be part of the string.

Related Functions

- UniStrrchr

Example

```
This example shows how to search a Unicode string for the first occurence
of a code element.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    ucs1[] = L"computer program";
UniChar    *puni;

UniChar    uni_char = L'p';
         puni = UniStrchr(ucs1, uni_char);
         if(puni) {
            printf("The character was found in the string\n");
            return (ULS_SUCCESS);
         }

}
```

UniStrcmp

UniStrcmp compares code element strings.

Format

```
#include <unidef.h>
```

int UniStrcmp
(const UniChar *ucs1, const UniChar *ucs2)

Parameters

ucs1 (const UniChar *)
String to be compared.

ucs2 (const UniChar *)
String to be compared.

Returns

return value (int) - returns
UniStrcmp returns an integer greater than, equal to, or less than zero, according to whether the code element string pointed to by ucs1 is greater than, equal to, or less than the code element string pointed to by ucs2.

Remarks

UniStrcmp compares the code element string pointed to by ucs1 to the code element string pointed to by ucs2.

Related Functions

- UniStrcmpi
- UniStrncmp
- Unistrncmpi

Example

```
This example shows how to compare Unicode strings.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    ucs1[] = L"computer";

UniChar    ucs2[] = L"program";

int          result = 0;
         result = UniStrcmp(ucs1, ucs2);
         if ( result == 0 )
             printf("The strings are identical\n");
         else
             printf("The strings are not identical\n");
         return (ULS_SUCCESS);

}
```

UniStrcmpi

UniStrcmpi compares strings without sensitivity to case.

Format

```
#include <unidef.h>
```

int UniStrcmpi
(const LocaleObject locale_object, const UniChar *ucs1, const UniChar *ucs2)

Parameters

locale_object (const LocaleObject)
A valid locale object created by a call to UniCreateLocaleObject or NULL

ucs1 (const UniChar *)
A null terminated UniChar string to be compared.

ucs2 (const UniChar *)
A null terminated UniChar string to be compared.

Returns

return value (int) - returns

less than 0
ucs1 less than ucs2

0
ucs1 equivalent to ucs2

Greater than 0
ucs1 greater than ucs

Remarks

UniStrcmpi compares ucs1 and ucs2 without sensitivity to case. All characters are converted to lowercase before the comparison. The locale object is used to convert the characters to lowercase. The locale may be specified as NULL to indicate default Unicode casing.

Related Functions

- UniStrcmp
- UniStrncmp
- UniStrncmpi

Example

```
This example shows how to compare Unicode strings without sensitivity to case.
#include <stdio.h>
```

```
#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;
UniChar      ucs1[] = L"computer";
UniChar      ucs2[] = L"COMPUTER";
int          result = 0;
int          rc = ULS_SUCCESS;
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }

         result = UniStrcmpi(locale_object, ucs1, ucs2);

         if ( result == 0 )
             printf("The strings are identical\n");
         else
             printf("The strings are not identical\n");
         return (ULS_SUCCESS);

}
```

UniStrcoll

UniStrcoll compares language collation strings.

Format

```
#include <unidef.h>
```

int UniStrcoll
    (const LocaleObject locale_object, const UniChar *ucs1, const UniChar *ucs2)

Parameters

locale_object (const LocaleObject)
    A locale object created by UniCreateLocaleObject or NULL.

ucs1 (const UniChar *)
    A string to be compared.

ucs2 (const UniChar *)
    A string to be compared.

Returns

return value (int) - returns

UniStrcoll returns an integer greater than, equal to, or less than zero. The integer returned depends on whether:

- The string pointed to by ucs1 is greater than, equal to, or less than the character string pointed to by ucs2
- Both character strings are interpreted as appropriate to the LC_COLLATE category of the locale indicated by the locale-object-handle argument, locale_object. The locale may be specified as NULL to indicate default Unicode collation.

Remarks

UniStrcoll compares the string pointed to by ucs1 to the string pointed to by ucs2, both interpreted as appropriate to the LC_COLLATE category of the locale indicated by the locale object handle argument, locale_object.

Example

```
This example shows how to compare Unicode strings using the collating
sequence specified by the locale object.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;
UniChar      ucs1[] = L"axe";
UniChar      ucs2[] = L"ant";
int          result = 0;
int          rc = ULS_SUCCESS;
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
```

result = UniStrcoll(locale_object, ucs1, ucs2);

```
         if ( result == 0 )
             printf("The strings are identical\n");
         else if ( result < 0 )
             printf("String1 is less than String2\n");
         else
             printf("String1 is greater than String2\n");
         return (ULS_SUCCESS);
```

}

UniStrcpy

UniStrcpy copies code element string.

Format

```
#include <unidef.h>
```

UniChar * UniStrcpy
(UniChar *ucs1, const UniChar *ucs2)

Parameters

ucs1 (UniChar *)
Target string.

ucs2 (const UniChar *)
Source string.

Returns

return value (UniChar *) - returns
UniStrcpy returns ucs1.

Remarks

UniStrcpy copies the code element string pointed to by ucs2 (including the terminating null code element) into the code element array pointed to by ucs1. If copying takes place between objects that overlap, the results are unpredictable. **Related Functions**

- UniStrncpy

Example

```
This example shows how to copy Unicode strings.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    ucs1[] = L"computer";

UniChar    ucs2[10];

UniChar    *puni;
         puni = UniStrcpy(ucs2, ucs1);
}
```

UniStrcspn

UniStrcspn searches for code element string.

Format

```
#include <unidef.h>
```

size_t UniStrcspn
(const UniChar *ucs1, const UniChar *ucs2)

Parameters

ucs1 (const UniChar *)
String to be searched.

ucs2 (const UniChar *)
String to search for.

Returns

return value (size_t) - returns
UniStrcspn returns the length of the segment.

Remarks

UniStrcspn computes the length of the maximum initial segment of the code element string, pointed to by ucs1, which consists entirely of code elements not from the string pointed to by ucs2.

Related Functions

- UniStrspn

Example

```
This example shows how to search Unicode strings.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    ucs1[] = L"This is the source string";
UniChar    ucs2[] = L"source";

size_t     result;
         result = UniStrcspn(ucs1, ucs2);
         return (ULS_SUCCESS);
}
```

UniStrfmon

UniStrfmon converts monetary value to string.

Format

```
#include <unidef.h>
```

int UniStrfmon
(const LocaleObject locale_object, UniChar *ucs, size_t maxsize, const UniChar *format, ...)

Parameters

locale_object (const LocaleObject)
A locale object created by UniCreateLocaleObject or NULL.

ucs (UniChar *)
Target string.

maxsize (size_t)
Maximum number of code elements to be placed in the target string.

format (const UniChar *)
Format to be used when formulating the target string.

... ( )
Zero or more arguments fetched according to the format string.

Returns

return value (int) - returns
If the total number of code elements, including the null terminating code element, is not more than maxsize, UniStrfmon returns the number of code elements placed into the array pointed to by ucs, not including the null terminating code element.

Remarks

The locale may be specified as NULL to indicate C locale.

UniStrfmon places characters into the array pointed to by ucs as controlled by the string pointed to by format. No more than maxsize code elements are placed into the array. The character string format contains two types of objects: plain characters, which are copied to the output stream, and directives, each of which results in the fetching of zero or more converted and formatted arguments. The results are unpredictable if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are ignored. A directive consists of a % character, optional conversion specifications, and a terminating character that determines the directive's behavior.

UniStrfmon converts numeric values to monetary strings, according to the specifications in the format parameter. This parameter also contains numeric values to be converted. Characters are placed into this ucs array, as controlled by the format parameter. The LC_MONETARY category governs the format of the conversion.

UniStrfmon can be called multiple times by including additional format structures, as specified by the format parameter.

The format parameter specifies a character string that can contain plain characters and conversion specifications. Plain characters are copied to the output stream. Conversion specifications result in the fetching of zero or more arguments, which are converted and formatted.

If there are insufficient arguments for the format parameter, the results are undefined. If arguments remain after the format parameter is exhausted, the excess arguments are ignored.

A conversion specification sequence consists of a % (percent) sign, optional flags, optional field width, optional left precision, optional right precision, and a required conversion character that determine the conversion to be performed.

One or more of the following flags can be specified to control the conversion:

=f
: An = (equals sign) followed by a single character *f* that specifies the numeric fill character. The default numeric fill character is the space character. This flag does not affect field width filling, which always uses the space character. This flag is ignored unless a left precision is specified. The fill character must be representable in a single byte in order to work with precision and width counts.

^
: Do not format the currency amount with grouping characters. The default is to insert grouping characters if they are defined for the current locale.

+or(
: Determines the representation of positive and negative currency amounts. Only one of these flags may be specified. The locale's equivalent of + and - are used if + is specified. The locale's equivalent of enclosing negative amounts within parentheses is used if (is specified. If neither flag is included, the + style is used.

!
: Suppresses the currency symbol from the output conversion.

-
: Specifies the alignment. If this flag is present, all fields are left-justified (padded to the right) rather than right-justified.

FIELD WIDTH

w
: The decimal-digit string, w, specifies the minimum field width in which the result of the conversion is right-justified (or left-justified if the - flag is specified). The default is zero.

LEFT PRECISION

#n
: A # (pound sign) followed by a decimal-digit string, n, specifies the maximum number of digits to be formatted to the left of the radix character. This option can be specified to keep formatted output from multiple calls to UniStrfmon aligned in the same columns. It can also be used to fill unused positions with a special character (for example, $***123.45). This option causes an amount to be formatted as if it has the number of digits specified by the n variable. If more than n digit positions are required, this option is ignored. Digit positions in excess of those required are filled with the numeric fill character set with the =f flag.

  If defined for the current locale and not suppressed with the ^ flag, grouping separators are inserted before the fill characters (if any) are added. Grouping separators are not applied to fill characters even if the fill character is a digit.

RIGHT PRECISION

.p

A .(period) followed by a decimal-digit string, p, specifies the number of digits after the radix character. If the value of the p variable is 0, no radix character is used. If a right precision is not specified, a default specified by the current locale is used. The amount being formatted is rounded to the specified number of digits prior to formatting.

CONVERSION CHARACTERS

i

The double argument is formatted according to the current locale's international currency format; for example, in the U.S.: USD 1,234.56.

n

The double argument is formatted according to the current locale's national currency format; for example, in the U.S.: $1,234.56.

%

Convert to a %; no argument is converted. The entire conversion specification must be %%.

Related Functions

- UniStrftime

Example

```
This example shows how to convert a monetary value to a Unicode string using
the specified locale object.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;
UniChar      ucs[20];
int          max_size = 20;
int          elements;
int          rc = ULS_SUCCESS;
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /* elements contains number of code elements placed into ucs */
         elements = UniStrfmon(locale_object, ucs,
                               max_size, (UniChar *)L"%n", 123.45);
         return (ULS_SUCCESS);

}
```

UniStrftime

UniStrftime formats date and time.

Format

```
#include <unidef.h>
#include <time.h>
```

size_t UniStrftime
(const localeObject locale_object, UniChar *ucs, size_t maxsize, const UniChar *format, const struct tm *timeptr)

Parameters

locale_object (const localeObject)
A locale object created by UniCreateLocaleObject

ucs (UniChar *)
Target string.

maxsize (size_t)
Maximum number of characters to be placed in ucs.

format (const UniChar *)
Format of the target string.

timeptr (const struct tm *)
Structure containing time and date information.

Returns

return value (size_t) - returns
If the total number of resulting code elements, including the null code element, is not more than maxsize, the number of code elements placed into the memory location addressed by ucs (not including the null code element) is returned. Otherwise, zero is returned and the contents of the memory location are indeterminate.

Remarks

Convert the internal time and date specification into a character string and place the results in the area pointed to by ucs under the direction of format. The null-terminated result of, at most, maxsize code elements, is placed in the memory location addressed by ucs. The format string may contain conversion specification characters and characters that are placed unchanged into the array. The characters that are converted are determined by the LC_TIME category of the locale indicated by the locale object handle argument locale_object and by the values in the time structure pointed to by timeptr. The results are unpredictable when objects being copied overlap.

The **format** parameter is a character string containing two types of objects: plain characters that are simply placed in the output string and conversion specifications that convert information from the **timeptr** parameter into readable form in the output string.

A % (percent sign) introduces a conversion specification.

The type of conversion is specified by one or two conversion characters. The characters and their meanings are:

%a
Represents the locale's abbreviated weekday name (for example, Sun).

%A
Represents the locale's full weekday name (for example, Sunday).

%b
Represents the locale's abbreviated month name (for example, Jan).

%B
Represents the locale's full month name (for example, January).

%c
Represents the locale's date and time format.

%C
Represents the century as a decimal number (00-99).

%d
Represents the day of the month as a decimal number (01 to 31).

%D
Represents the date in %m/%d/%y format (for example, 01/31/94).

%e
Represents the day of the month as a decimal number (1 to 31). A single digit is preceded by a space character.

%h
Same as %b.

%H
Represents the 24-hour-clock hour as a decimal number (00 to 23).

%I
Represents the 12-hour-clock hour as a decimal number (01 to 12).

%j
Represents the day of the year as a decimal number (001 to 366).

%m
Represents the month of the year as a decimal number (01 to 12).

%M
Represents the minute of the hour as a decimal number (00 to 59).

%n
Specifies a new-line character.

%p
Represents the locale's AM or PM string.

%r
Represents the time with AM/PM notation (%I:%M:%S%p).

%R

Represents 24-hour-clock time in the format %H:%M.

%S
: Represents the second of the minute as a decimal number (00 to 61).

%t
: Specifies a tab character.

%T
: Represents the time in the format %H:%M:%S.

%u
: Represents the day of the week as a decimal number (1 to 7). 1 represents Monday.

%U
: Represents the week of the year as a decimal number (00 to 53). Sunday is considered the first day of the week.

%V
: Represents the week of the year as a decimal number (01 to 53). Monday is considered the first day of the week. If the week containing 1 January has four or more days in the new year, then it is considered week 1; otherwise, it is week 53 of the previous year, and the next week is week 1.

%w
: Represents the day of the week as a decimal number (0 to 6). 0 represents Sunday.

%W
: Represents the week of the year as a decimal number (00 to 53). Monday is considered the first day of the week. All days in a new year preceding the first Sunday are considered to be week 0.

%x
: Represents the locale's date format.

%X
: Represents the locale's time format.

%y
: Represents the year of the century (00 to 99).

%Y
: Represents the year with century as a decimal number for example (1994).

%Z
: Represents the time-zone name or abbreviation if one can be determined (for example EST). Replaced by no bytes if time zone information does not exist.

%%
: Specifies a % (percent) sign.

Some conversion specifiers can be modified by the E or O modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist for the current locale, the behavior will be as if the unmodified conversion specification were used.

%Ec
: Locale's alternative appropriate date and time representation.

%EC
: is the name of the base year (period) in the locale's alternative representation.

%Ex
: is the locale's alternative date representation.

%EX
: is the locale's alternative time representation.

%Ey
: is the offset from %EC (year only) in the locale's alternative representation.

%EY
: is the full alternative year representation.

%Od
: is the day of the month using the locale's alternative numeric symbols; filled as needed with leading zeros if there is any alternative symbol for zero, otherwise with leading spaces.

%Oe
: is replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading spaces.

%OH
: is the hour (24-hour clock) using the locale's alternative numeric symbols.

%OI
: is the hour (12-hour clock) using the locale's alternative numeric symbols.

%Om
: is the month using the locale's alternative numeric symbols.

%OM
: is the minutes using the locale's alternative numeric symbols.

%OS
: is the seconds using the locale's alternative numeric symbols.

%Ou
: is the weekday of the year (Monday = 1) using the locale's alternative numeric symbols.

%OU
: is the week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.

%OV
: is replaced by the week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.

%Ow
: is the number of the weekday (Sunday=0) using the locale's alternative numeric symbols.

%OW
: is the week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.

%Oy
: is the year (offset from %C) in the locale's alternative representation and using the locale's alternative numeric symbols.

Related Functions

- UniStrfmon
- UniStrptime

Example

```
This example shows how to convert a date and time to a Unicode string using
the specified locale object.
#include <stdio.h>

#include <time.h>
```

```
#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;

struct tm    *ptime;
time_t       time_date;
UniChar      ucs[30];
int          max_size = 30;
int          elements;
int          rc = ULS_SUCCESS;
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         time_date = time(NULL);
         ptime = localtime(&time_date);
         /* elements contains number of code elements placed into ucs */
         elements = UniStrftime(locale_object, ucs,
                                max_size, (UniChar *)L"%a %b %d\n %I:%M %p",
                                ptime);
         return (ULS_SUCCESS);

}
```

UniStrlen

UniStrlen determines code element count.

Format

```
#include <unidef.h>
```

size_t UniStrlen
    (const UniChar *ucs)

Parameters

ucs (const UniChar *)
    A null-terminated string composed of UniChar code elements.

Returns

return value (size_t) - returns
    UniStrlen returns the number of code elements that precede the terminating null code element.

Remarks

UniStrlen computes the length of the code element string pointed to by ucs.

Example

```
This example shows how to determine the code element count of a Unicode string.
#include <stdio.h>

#include <unidef.h>
int main(void) {
int   elements;
         /* elements contains number of code elements in the Unicode string */
         elements = UniStrlen((UniChar *)L"program");
         return (ULS_SUCCESS);
}
```

UniStrlwr

UniStrlwr converts a Unicode string to lowercase according to the language neutral case mapping tables.

Format

```
#include <unidef.h>
```

UniChar *UniStrlwr
(UniChar *ucsUniStringIn)

Parameters

ucsUniStringIn (UniChar *) - input
Unicode string to be mapped to lowercase.

Returns

ucsUniStringOut (UniChar *) - returns
Converted lowercase string.

Remarks

The input string must be null-terminated.

Related Functions

- UniStrupr
- UniTolower
- UniToupper

Example

```
This example shows how to convert a Unicode string to lowercase.
```

```
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar   *plwr_unistr;
         /* plwr_unistr points to the converted lowercase Unicode string */
         plwr_unistr = UniStrlwr((UniChar *)L"IBM");
         return (ULS_SUCCESS);
}
```

UniStrncat

UniStrncat concatenates a specific number of code elements.

Format

```
#include <unidef.h>
```

UniChar * UniStrncat
(UniChar *ucs1, const UniChar *ucs2, size_t n)

Parameters

ucs1 (UniChar *)
String to be appended to.

ucs2 (const UniChar *)
String to concatenate.

n (size_t)
Number of elements in ucs2 to concatenate.

Returns

return value (UniChar *) - returns
Concatenated string.

Remarks

UniStrncat appends not more than n code elements (a null code element and code elements that follow it are not appended) from the code element array pointed to by ucs2 to the end of the code element string pointed to by ucs1. The initial code element of ucs2 overwrites the null code element at the end of ucs1. A terminating null code element is always appended to the result. If copying takes place between objects that overlap, the results are unpredictable.

Related Functions

- UniStrcat

Example

```
This example shows how to concatenate a specific number of code
elements onto a Unicode string.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    ucs1[40] = L"computer";

size_t     num_elems = 3;
         UniStrncat(ucs1, (UniChar *)L" program", num_elems);
         return (ULS_SUCCESS);
}
```

UniStrncmp

UniStrncmp compares a specific number of code elements.

Format

```
#include <unidef.h>
```

int UniStrncmp
(const UniChar *ucs1, const UniChar *ucs2, size_t n)

Parameters

ucs1 (const UniChar *)
String to compare.

ucs2 (const UniChar *)
String to compare.

n (size_t)
Number of code elements to compare.

Returns

return value (int) - returns
UniStrncmp returns an integer greater than, equal to, or less than zero. The integer returned depends on whether the possibly null-terminated code element array pointed to by ucs1 is greater than, equal to, or less than the possibly null-terminated code element array pointed to by ucs2.

Remarks

UniStrncmp compares not more than *n* code elements (code elements that follow a NULL code element are not compared) from the code element array pointed to by ucs1 to the code element array pointed to by ucs2.

Related Functions

- UniStrcmp
- UniStrcmpi
- UniStrncmpi

Example

```
This example shows how to compare a specific number of code
elements.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    ucs1[] = L"computer";

UniChar    ucs2[] = L"computer program";
size_t     num_elems = 3;
int        result = 0;

         result = UniStrncmp(ucs1, ucs2, num_elems);
         if ( result == 0 )
             printf("The strings are identical\n");
         else
             printf("The strings are not identical\n");
         return (ULS_SUCCESS);

}
```

UniStrncmpi

UniStrncmpi compares one or more code elements of strings without sensitivity to case.

Format

```
#include <unidef.h>
```

int UniStrncmpi
    (const LocaleObject locale_object, const UniChar *ucs1, const UniChar *ucs2, const size_t n)

Parameters

locale_object (const LocaleObject)
    A valid locale object created by a call to UniCreateLocaleObject or NULL.
ucs1 (const UniChar *)
    A null-terminated UniChar string to be compared.
ucs2 (const UniChar *)
    A null terminated UniChar string to be compared.

n (const size_t)
: The maximum number of code elements to compare.

Returns

return value (int) - returns

less than 0
: ucs1 less than ucs2

0
: ucs1 equivalent to ucs2

Greater than 0
: ucs1 greater than ucs

Remarks

UniStrncmpi compares ucs1 and ucs2 without sensitivity to case. All *n* code elements are converted to lowercase before the comparison. The locale object is used to convert the characters to lowercase. A maximum of *n* code elements are compared. The locale may be specified as NULL to indicate Unicode casing.

Related Functions

- UniStrcmp
- UniStrcmpi
- UniStrncmp

Example

```
This example shows how to compare a specific number of code
elements without sensitivity to case.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject  locale_object;
UniChar       ucs1[] = L"COMPUTER";
UniChar       ucs2[] = L"computer program";
size_t        num_elems = 3;
int           result = 0;
int           rc;
         /*************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                              */
         /*************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         result = UniStrncmpi(locale_object, ucs1, ucs2, num_elems);
         if ( result == 0 )
             printf("The strings are identical\n");
         else
             printf("The strings are not identical\n");
```

```
        return (ULS_SUCCESS);

}
```

UniStrncpy

UniStrncpy copies a specific number of code elements.

Format

```
#include <unidef.h>
```

UniChar * UniStrncpy
(UniChar *ucs1, const UniChar *ucs2, size_t n)

Parameters

ucs1 (UniChar *)
Target string.

ucs2 (const UniChar *)
Source string.

n (size_t)
Number of elements in ucs2 to copy.

Returns

return value (UniChar *) - returns
UniStrncpy returns ucs1.

Remarks

UniStrncpy copies not more than n code elements (code elements that follow a null code element are not copied) from the code element array pointed to by ucs2 to the code element array pointed to by ucs1. If copying takes place between objects that overlap, the results are unpredictable. If the code element array pointed to by ucs2 is a code element string that is shorter than *n* code elements, null code elements are appended to the copy in the code element array pointed to by ucs1, until *n* code elements, in all, have been written.

Related Functions

- UniStrcpy

Example

```
This example shows how to copy a specific number of code
elements in a Unicode string.
```

```
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    ucs1[] = L"computer";

UniChar    ucs2[10];

UniChar    *puni;

size_t     num_elems = 4;
         puni = UniStrncpy(ucs2, ucs1, num_elems);
         return (ULS_SUCCESS);
}
```

UniStrpbrk

UniStrpbrk locates code elements in a code element string.

Format

```
#include <unidef.h>
```

UniChar * UniStrpbrk
(const UniChar *ucs1, const UniChar *ucs2)

Parameters

ucs1 (const UniChar *)
String to be searched.

ucs2 (const UniChar *)
String to search for.

Returns

return value (UniChar *) - returns
UniStrpbrk returns a pointer to the code element or a null pointer, if the code element from the code element string from ucs2, does not occur in ucs1.

Remarks

UniStrpbrk locates the first occurrence, in the string pointed to by ucs1, of any code element from the code element string pointed to by ucs2. **Related Functions**

- UniStrstr

Example

```
This example shows how to locate code elements in a Unicode string.
```

```
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    ucs1[] = L"computer";

UniChar    ucs2[] = L"put";

UniChar    *puni;
         puni = UniStrpbrk(ucs1, ucs2);
         if (puni)
             printf("The sub string was found\n");
         return (ULS_SUCCESS);

}
```

UniStrptime

UniStrptime converts date and time.

Format

```
#include <unidef.h>
#include <time.h>
```

UniChar * UniStrptime
(const LocaleObject locale_object, const UniChar *buf, const UniChar *fmt, struct tm *tm)

Parameters

locale_object (const LocaleObject)
  A locale object created by UniCreateLocaleObject.

buf (const UniChar *)
  String to be converted.

fmt (const UniChar *)
  Format of the source string.

tm (struct tm *)
  Time/data structure to receive the time/data information held in the string buf.

Returns

return value (UniChar *) - returns
  Returns a pointer to the character following the last character; otherwise, a NULL pointer is returned and the contents of the tm structure are undefined.

Remarks

UniStrptime converts the character string pointed to by buf to a time value, which is stored in the structure pointed to by tm, using the format specified by fmt. A pointer to the character following the last character in the string pointed to by buf is returned. The character string pointed to by fmt consists of field descriptors and text characters, similar to the scanf. Each field descriptor consists of a % character followed by another character that specifies the replacement for the field descriptor. The type of conversion is specified by one or two conversion characters. The characters and their meanings are specified in the **Format Strings** and **Modified Directives** sections.

FORMAT STRINGS

%a
: Day of the week, abbreviated or full name may be specified (for example Sun).

%A
: Same as %a.

%b
: Represents the locale's month name, abbreviated or fullname may be specified.

%B
: Same as %b.

%c
: Represents the locale's date and time format.

%C
: Represents the century number (0 to 99).

%d
: Represents the day of the month as a decimal number (01 to 31).

%D
: Represents the date in %m/%d/%y format (for example, 01/31/91).

%e
: Same as %d.

%h
: Same as %b.

%H
: Represents the 24-hour-clock hour as a decimal number (00 to 23).

%I
: Represents the 12-hour-clock hour as a decimal number (01 to 12).

%j
: Represents the day of the year as a decimal number (001 to 366).

%m
: Represents the month of the year as a decimal number (01 to 12).

%M
: Represents the minute of the hour as a decimal number (00 to 59).

%n
: Represents any white space.

%p
: Represents the locale's AM or PM string.

%r
: Represents the time as %I:%M:%S%p.

%R
: Represents the time as %H:%M.

%S
: Represents the second of the minute as a decimal number (00 to 61).

%t
: Represents any white space.

%T
: Represents time in the format %H:%M:%S.

%U
: Represents the week of the year as a decimal number (00 to 53). Sunday is considered the first day of the week.

%w
: Represents the day of the week as a decimal number (0 to 6). Sunday is considered as 0.

%W
: Represents the week of the year as a decimal number (00 to 53). Monday is considered the first day of the week.

%x
: Represents the locale's date format.

%X
: Represents the locale's time format.

%y
: Represents the year of the century (00 to 99).

%Y
: Represents the year with century as a decimal number (for example 1994).

%%
: Specifies a % (percent) sign.

MODIFIED DIRECTIVES

Some directives can be modified by the E and O modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified directive. If the alternative format or specification does not exist in the current locale, the behavior will be as if the unmodified directive were used.

%Ec
: is the locale's alternative appropriate date and time representation.

%EC
: is the name of the base year (period) in the locale's alternative representation.

%Ex
: is the locale's alternative date representation.

%EX
: is the locale's alternative time representation.

%Ey
: is the offset from %EC (year only) in the locale's alternative representation.

%EY
: is the full alternative year representation.

%Od
: is the day of the month using the locale's alternative numeric symbols; leading zeros are permitted but not required.

%Oe

is the same as %Od.

%OH
: is the hour (24-hour clock) using the locale's alternative numeric symbols.

%OI
: is the hour (12-hour clock) using the locale's alternative numeric symbols.

%Om
: is the month using the locale's alternative numeric symbols.

%OM
: is the minutes using the locale's alternative numeric symbols.

%OS
: is the seconds using the locale's alternative numeric symbols.

%OU
: is the week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.

%Ow
: is the number of the weekday (Sunday=0) using the locale's alternative numeric symbols.

%OW
: is the week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.

%Oy
: is the year (offset from %C) in the locale's alternative representation and using the locale's alternative numeric symbols.

A format specification consisting of white-space characters is performed by reading input until the first nonwhite-space character (which is not read) or no more characters can be read.

A format specification consisting of an ordinary character is performed by reading the next character from the string parameter. If this character differs from the character comprising the directive, the directive fails and the differing character and any characters following it remain unread. Case is ignored when matching string items, such as month or weekday names.

Related Functions

- UniStrftime

Example

```
This example shows how to convert a time date string to a time structure.
#include <stdio.h>

#include <time.h>

#include <unidef.h>
int main(void) {
LocaleObject  locale_object;
UniChar       uni_fmt[] = L"%A %b %d %r %Y";
UniChar       uni_time_str[] = L"Wednesday Oct 23 03:07:00 PM 1995";
UniChar       *puni;
struct tm     convrt_time;
int           rc;
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"en_US", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         puni = UniStrptime(locale_object, uni_time_str, uni_fmt,
                            &convrt_time);
         if ( puni == NULL ) {
             printf("UniStrptime error\n");
             return (1);
         }
         else
             return (ULS_SUCCESS);

}
```

UniStrrchr

UniStrrchr locates last occurrence of code element.

Format

```
#include <unidef.h>
```

UniChar * UniStrrchr
(const UniChar *ucs, UniChar uc)

Parameters

ucs (const UniChar *)
String to be searched.

uc (UniChar)
UniChar code element to search for.

Returns

return value (UniChar *) - returns
UniStrrchr returns either a pointer to the found code element or a null pointer, if uc does not occur in the code element string.

Remarks

UniStrrchr locates the last occurrence of uc in the code element string pointed to by ucs. The terminating null code element is considered to be part of the string.

Related Functions

- UniStrchr

Example

```
This example shows how to locate the last occurrence of a code element
in a Unicode string.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    ucs[] = L"computer";

UniChar    uc = L't';

UniChar    *puni;
         puni = UniStrrchr(ucs, uc);
         if (puni)
             printf("The character is contained in the string\n");
         return (ULS_SUCCESS);

}
```

UniStrspn

UniStrspn determines the number of code elements in a segment.

Format

```
#include <unidef.h>
```

size_t UniStrspn
(const UniChar *ucs1, const UniChar *ucs2)

Parameters

ucs1 (const UniChar *)
String to be searched.

ucs2 (const UniChar *)
String of code elements to search for.

Returns

return value (size_t) - returns
UniStrspn returns the length of the segment.

Remarks

UniStrspn computes the length of the maximum initial segment of the code element string pointed to by ucs1, which consists entirely of code elements from the code element string pointed to by ucs2.

Related Functions

- UniStrcspn

Example

```
This example shows how to determine the number of elements in a Unicode string.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    ucs1[] = L"computer";

UniChar    ucs2[] = L"omc";

size_t     num_elems = 0;
         num_elems = UniStrspn(ucs1, ucs2);
         if (num_elems)
             printf("The first %d characters were found\n", num_elems);
         return (ULS_SUCCESS);

}
```

UniStrstr

UniStrstr locates a code element sequence.

Format

```
#include <unidef.h>
```

UniChar * UniStrstr
(const UniChar *ucs1, const UniChar *ucs2)

Parameters

ucs1 (const UniChar *)
String to be searched.
ucs2 (const UniChar *)
String of code elements to search for.

Returns

return value (UniChar *) - returns
UniStrstr returns either a pointer to the located code element string or a null pointer, if the string is not found.

If ucs2 points to a code element string with zero length, the function returns ucs1.

Remarks

UniStrstr locates the first occurrence, in the code element string pointed to by ucs1 of the sequence of code elements (excluding the ending null code element), in the code element string pointed to by ucs2.

Related Functions

- UniStrpbrk

Example

```
This example shows how to locate a code element sequence in a Unicode string.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    ucs1[] = L"computer";

UniChar    ucs2[] = L"put";

UniChar    *puni;
         puni = UniStrstr(ucs1, ucs2);
         if (puni)
             printf("The code element sequence was found\n");
         return (ULS_SUCCESS);

}
```

UniStrtod

UniStrtod converts character string to double-precision floating point.

Format

```
#include <unidef.h>
```

int UniStrtod
(const LocaleObject locale_object, const UniChar *nptr, UniChar **endptr, double *result)

Parameters

locale_object (const LocaleObject)
: A locale object created by UniCreateObject.

nptr (const UniChar *)
: String to be converted.

endptr (UniChar **)
: A pointer to the first UniChar that is not recognized as being part of a number.

result (double *)
: Resulting double-precision floating-point number.

Returns

return value (int) - returns

UniStrtod returns one of the following values:

ULS_SUCCESS
: The function was successful.

ULS_BADOBJ
: Invalid locale object specified.

ULS_INVALID
: The endptr or result argument contains an invalid pointer value. The reliable detection of this error is implementation dependent.

ULS_RANGE
: The conversion resulted in an out-of-range condition.

Remarks

UniStrtod converts the initial portion of the string pointed to by nptr to double-precision floating-point representation. First, it decomposes the input string into three parts:

1. An initial, possibly empty, sequence of white-space characters (as indicated by the space attribute).
2. A subject sequence resembling a floating-point constant.
3. A final string of one or more unrecognized characters, including the terminating null character of the input string.

Then, it attempts to convert the subject sequence to a floating-point number, and returns the result in the area pointed to by result. A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

Related Functions

- UniStrtol
- UniStrtok
- UniStrtoul

Example

```
This example shows how to convert a Unicode string to a double precision
floating point number.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject  locale_object;
UniChar       uni_string[] = L"3.1415926This stopped it";
UniChar       *uni_stop_string;
double        double_num;
int           rc;
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                   (UniChar *)L"", &locale_object);
```

```
          if (rc != ULS_SUCCESS) {
            printf("UniCreateLocaleObject error: return code = %u\n", rc);
            return 1;
          }
          rc = UniStrtod(locale_object, uni_string, &uni_stop_string,
                         &double_num);
          if (rc != ULS_SUCCESS) {
            printf("UniStrtod error: return code = %u\n", rc);
            return 1;
          }
          else {
            printf("The double precision number is %f\n", double_num);
            return (ULS_SUCCESS);
          }

}
```

UniStrtol

UniStrtol converts a character string to a long integer.

Format

```
#include <unidef.h>
```

int UniStrtol
(const LocaleObject locale_object, const UniChar *nptr, UniChar **endptr, int base, long int *result)

Parameters

locale_object (const LocaleObject)
    A locale object created by UniCreateLocaleObject or NULL.

nptr (const UniChar *)
    String to be converted.

endptr (UniChar **)
    A pointer to the first UniChar that is not recognized as being part of a number.

base (int)
    The radix used to perform conversion.

result (long int *)
    The resulting integer.

Returns

return value (int) - returns

UniStrtol returns one of the following values:

ULS_SUCCESS
: The function was successful.

ULS_BADOBJ
: Invalid locale object specified.

ULS_INVALID
: The endptr or result argument contains an invalid pointer value. The reliable detection of this error is implementation-dependent.

ULS_RANGE
: The conversion resulted in an out-of-range condition.

Remarks

The locale may be specified as NULL to indicate C locale.

UniStrtol converts the initial portion of the string pointed to by nptr to long int representation. First, it decomposes the input string into three parts:

1. An initial, possibly empty, sequence of white-space characters (as indicated by the space attribute).
2. A subject sequence, resembling an integer, that is represented in the radix and determined by the value of base.
3. A final string of one or more unrecognized characters, including the ending null character of the input string.

Then, it attempts to convert the subject sequence to an unsigned integer, and returns the result in the area pointed to by result. A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

If the value of base is between 2 and 36, the letters a (or A) to z (or Z), inclusive, are ascribed the values 10 to 35. Only letters whose ascribed value is less than base are permitted.

If base is set to 0, the expected form of the subject sequence is a decimal, octal, or hexadecimal constant. Decimal constants begin with a nonzero digit. Octal constants begin with 0. Hexadecimal constants begin with 0x or 0X.

Related Functions

- UniStrtod
- UniStrtok
- UniStrtoul

Example

```
This example shows how to convert a Unicode string to a long integer.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject  locale_object;
UniChar       uni_string[] = L"110134932";
UniChar       *uni_stop_string;
long int      long_num;
int           rc;
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                  */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                   (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         rc = UniStrtol(locale_object, uni_string, &uni_stop_string,
                        10, &long_num);
         if (rc != ULS_SUCCESS) {
           printf("UniStrtol error: return code = %u\n", rc);
           return 1;
         }
         else {
           printf("The long integer is %ld\n", long_num);
           return (ULS_SUCCESS);
         }

}
```

UniStrtok

UniStrtok converts a Unicode string to tokens.

Format

```
#include <unidef.h>
```

UniChar *UniStrtok
(UniChar *ucsString1, const UniChar *ucsString2)

Parameters

ucsString1 (UniChar *) - input
Unicode string containing zero or more tokens.

ucsString1 is a string of zero or more tokens. The tokens in ucsString1 can be separated by one or more of the delimiters in ucsString2. UniStrtok does not support the passing of NULL for ucsString1 parameter as is supported in the ANSI C strtok function.

ucsString2 (const UniChar *) - input
Set of UniChar characters that can be used as delimiters.

ucsString2 is the set of characters serving as delimiters of the tokens in ucsString1.

Returns

ucsToken (UniChar *) - returns
  Pointer to the first token.

Remarks

UniStrtok will return the first token in the string specified in ucsString1. UniStrtok replaces the delimiter character with 0x0000 and returns a pointer to the token.

Related Functions

- UniStrtod
- UniStrtol
- UniStrtoul

Example

```
This example shows how to convert a Unicode string to tokens.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    *uni_string = L"a string, of, ,tokens";
UniChar    *puni_token;
int         uni_len1;
int         uni_len2;
int         token_count = 0;
         uni_len1 = UniStrlen(uni_string);
         puni_token = UniStrtok(uni_string, (UniChar *)L",");
         ++token_count;
         /* Continue to loop through the string looking for tokens */
         do
         {
            uni_len2 = UniStrlen(puni_token) + 1;
            puni_token += uni_len2;
            if(puni_token < uni_string + uni_len1)
            {
               puni_token = UniStrtok(puni_token, (UniChar *)L",");
               ++token_count;
            }
            else
               break;
         } while (1);
         printf("%d tokens were found\n", token_count);
         return (ULS_SUCCESS);

}
```

UniStrtoul

UniStrtoul converts a character string to an unsigned long integer.

Format

```
#include <unidef.h>
```

int UniStrtoul
(const LocaleObject locale_object, const UniChar *nptr, UniChar **endptr, int base, unsigned long int *result)

Parameters

locale_object (const LocaleObject)
A locale object created by UniCreateLocaleObject or NULL.

nptr (const UniChar *)
String to be converted.

endptr (UniChar **)
A pointer to the first UniChar that is not recognized as being part of a number.

base (int)
The radix used to perform conversion.

result (unsigned long int *)
The resulting unsigned long integer.

Returns

return value (int) - returns

UniStrtoul returns one of the following values:

ULS_SUCCESS
The function was successful.
ULS_BADOBJ
Invalid locale object specified.
ULS_INVALID
The endptr or result argument contains an invalid pointer value. The reliable detection of this error is implementation dependent.
ULS_RANGE
The conversion resulted in an out-of-range condition.

Related Functions

- UniStrtod
- UniStrtol
- UniStrtok

Remarks

The locale may be specified as NULL to indicate C locale.

UniStrtoul converts the initial portion of the string pointed to by nptr to unsigned long int representation. First, it decomposes the input string into three parts:

1. An initial, possibly empty, sequence of white-space characters (as indicated by the space attribute).
2. A subject sequence resembling an unsigned integer represented in some radix determined by the value of base.
3. A final string of one or more unrecognized characters, including the terminating null character of the input string.

Then, it attempts to convert the subject sequence to an unsigned integer, and returns the result in the area pointed to by result. A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

If the value of base is between 2 and 36, the letters a (or A) to z (or Z) inclusive are ascribed the values 10 to 35. Only letters whose ascribed value is less than base are permitted.

If base is set to 0, the expected form of the subject sequence is a decimal, octal or hexadecimal constant. Decimal constants begin with a nonzero digit. Octal constants begin with 0. Hexadecimal constants begin with 0x or 0X.

Example

```
This example shows how to convert a Unicode string to an unsigned long integer.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject       locale_object;
UniChar            uni_string[] = L"110134932";
UniChar            *uni_stop_string;
unsigned long int  long_num;
int                rc;
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         rc = UniStrtoul(locale_object, uni_string, &uni_stop_string,
                        10, &long_num);
         if (rc != ULS_SUCCESS) {
           printf("UniStrtoul error: return code = %u\n", rc);
           return 1;
         }
         else {
           printf("The unsigned long integer is %lu\n", long_num);
           return (ULS_SUCCESS);
         }

}
```

UniStrupr

UniStrupr converts a Unicode string to uppercase according to the language neutral case mapping tables.

Format

```
#include <unidef.h>
```

UniChar *UniStrupr
    (UniChar *ucsUniStringIn)

Parameters

ucsUniStringIn (UniChar *) - input
    Unicode string to be mapped to uppercase.

Returns

ucsUniStringOut (UniChar *) - returns
    Converted uppercase string.

Remarks

The input string must be null-terminated.

Related Functions

- UniStrlwr
- UniTolower
- UniToupper

Example

```
This example shows how to uppercase Unicode strings according to the
language neutral case mapping tables.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    ucs[] = L"computer";

UniChar    *puni;
         puni = UniStrupr(ucs);
         return (ULS_SUCCESS);

          }
```

UniStrxfrm

UniStrxfrm transforms a character string into collating weights.

Format

```
#include <unidef.h>
```

size_t UniStrxfrm
    (const LocaleObject locale_object, UniChar *ucs1, const UniChar *ucs2, size_t n)

Parameters

locale_object (const LocaleObject)
    A locale object created by UniCreateLocaleObject or NULL.

ucs1 (UniChar *) - output
    Target transformed string.

ucs2 (const UniChar *) - input
    Source string to be transformed.

n (size_t) - input
    Maximum number of code elements to be placed in ucs1.

Returns

return value (size_t) - returns

UniStrxfrm returns the length of the transformed string (not including the terminating null code element). If the value returned is n or more, the contents of the array pointed to by ucs1 are indeterminate. If ucs1 is a null pointer, UniStrxfrm returns the number of elements required to contain the transformed character string.

Remarks

UniStrxfrm transforms the string pointed to by ucs2 to values that represent character collating weights and places the resulting string into the array pointed to by ucs1. The transformation is such that, if UniStrcmp is applied to two transformed strings, it returns a value greater than, equal to, or less than 0, corresponding to the result of UniStrcoll applied to the same two original strings. No more than n elements are placed into the resulting array pointed to by ucs1, including the terminating null code element. If n is zero, ucs1 is permitted to be a null pointer. If copying takes place between objects that overlap, the results are unpredictable.

UniStrxfrm is controlled by the LC_COLLATE category of the locale as indicated by the locale object handle argument, locale_object. The locale may be specified as NULL to indicate Unicode collation.

Example

```
This example shows how to collect character collating weights from Unicode
strings using the specified locale object.
#include <stdio.h>
```

```
#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;
UniChar      *pucs1;
UniChar      *pucs2 = L"computer";
int          num_elems = 8;
int          num_elems_trx = 0;
int          result = 0;
int          rc = ULS_SUCCESS;
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         /********************************************************************/
         /* Calculate the space needed for the collating weights             */
         /********************************************************************/
         num_elems = UniStrxfrm (locale_object, NULL, pucs2, 0);
```

pucs1 = (UniChar *) malloc((num_elems + 1) * sizeof(UniChar));

```
         if(!pucs1)
            return 1;
         /********************************************************************/
         /* Obtain the collating weights for the Unicode string.             */
         /* num_elems_trx should be less than num_elems                      */
         /********************************************************************/
         num_elems_trx = UniStrxfrm (locale_object, pucs1,
                                      pucs2, num_elems + 1);
         if(num_elems_trx >= (num_elems + 1)) {
           printf("UniStrxfrm error:\n");
           return 1;
         }
         return (ULS_SUCCESS);
```

}

UniTolower

UniTolower converts a Unicode character to lowercase according to the language neutral case mapping tables.

Format

```
#include <unidef.h>
```

**UniChar UniTolower
(UniChar ucUniCharIn)**

Parameters

ucUniCharIn (UniChar) - input
    Unicode character to be mapped to lowercase.

Returns

ucUniCharOut (UniChar) - returns
    Converted lowercase character.

Related Functions

- UniStrlwr
- UniStrupr
- UniToupper

Example

```
This example shows how to convert a Unicode character to lowercase according to the
language neutral case mapping tables.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    uni_upr = L'C';

UniChar    uni_lwr;
         uni_lwr = UniTolower(uni_upr);
         return (ULS_SUCCESS);
}
```

UniToupper

UniToupper converts a Unicode character to uppercase according to the language neutral case mapping tables.

Format

```
#include <unidef.h>
```

UniChar UniToupper
    (UniChar ucUniCharIn)

Parameters

ucUniCharIn (UniChar) - input
    Unicode character to be mapped to uppercase.

Returns

ucUniCharOut (UniChar) - returns
    Converted uppercase character.

Related Functions

- UniStrlwr
- UniStrupr
- UniTolower

Example

```
This example shows how to convert a Unicode character to uppercase according to the
language neutral case mapping tables.
#include <stdio.h>

#include <unidef.h>
int main(void) {
UniChar    uni_lwr = L'c';

UniChar    uni_upr;
         uni_upr = UniToupper(uni_lwr);
         return (ULS_SUCCESS);
}
```

UniTransformStr

UniTransformStr transforms strings according to a XformObject created by UniCreateTransformObject.

Format

```
#include <unidef.h>
```

int UniTransformStr
(XformObject xform_object, const UniChar *InpBuf, int *InpSize, UniChar *OutBuf, int *OutSize)

Parameters

xform_object (XformObject)
An *xform_object* created by UniCreateTransformObject.

InpBuf (const UniChar *)
String to be transformed.

InpSize (int *)
Number of code elements in InpBuf.

OutBuf (UniChar *)
Target string.

OutSize (int *)

Number of code elements that OutBuf can hold.

Returns

return value (int) - returns

UniTransformStr returns one of the following:

ULS_SUCCESS
Transformation completed without errors.

ULS_UNSUPPORTED
The transform object was not found.

Remarks

UniTransformStr transforms a UniChar character string as specified by the transformation object handle xform_object for the LC_CTYPE category. This category applies to the locale object that was used to create the transformation handle xform_object (by UniCreateTransformObject). The text from the input buffer is transformed and the result is placed in the output buffer. Any characters not included in the transformation type referenced by xform_object are moved, to the output buffer, unchanged.

The InpSize argument, on input, specifies the number of code elements to be transformed. A value of -1 indicates that the input is delimited by a UniChar NULL character (0x0000). On return, the value is modified to the actual number of code elements processed in the source string.

The OutSize argument, on input, specifies the size of the output buffer (number of code elements). On return, the value is modified to the actual number of code elements placed in OutBuf.

Related Functions

- UniCreateTransformObject
- UniFreeTransformObject

Example

```
This example shows how to create and use a transform object.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject locale_object = NULL;

XformObject  xform_object = NULL;
int          rc = ULS_SUCCESS;
int          in_unistr_elem = 0;
int          out_unistr_elem = 10;
UniChar      *pin_unistr = (UniChar *)L"os2";
UniChar      out_unistr[10];
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                 */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                    (UniChar *)L"", &locale_object);
```

```
        if (rc != ULS_SUCCESS) {
          printf("UniCreateLocaleObject error: return code = %u\n", rc);
          return 1;
        }
        /* Create an upper case transform object */
        rc = UniCreateTransformObject(locale_object,
                                      (UniChar *)L"upper", &xform_object);
        if (rc != ULS_SUCCESS) {
          printf("UniCreateTransformObject error: return code = %u\n", rc);
          return 1;
        }
        /* Calculate the number of elements to transform */
        in_unistr_elem = UniStrlen (pin_unistr) + 1;
        /* Make call to transform input string to uppercase */
        rc = UniTransformStr(xform_object, pin_unistr,
                             &in_unistr_elem, out_unistr,
                             &out_unistr_elem);
        if (rc != ULS_SUCCESS) {
          printf("UniTransformStr error: return code = %u\n", rc);
          return 1;
        }
        return ULS_SUCCESS;

}
```

UniTransLower

UniTransLower converts a Unicode character to lowercase using the specified locale.

Format

```
#include <unidef.h>
```

UniChar UniTransLower
(const LocaleObject locale_object, UniChar uc)

Parameters

locale_object (const LocaleObject)
A locale object created by UniCreateLocaleObject or NULL.

uc (UniChar)
The character to be transformed.

Returns

return value (UniChar) - returns
UniTransLower returns the transformed character. The input character is returned if there is no transformation defined for the character in locale_object.

Related Functions

- UniTransUpper

Example

```
This example shows how to convert a Unicode character to lowercase.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject  locale_object;
UniChar       uni_upr = L'C';
UniChar       uni_lwr;
int           rc;
         /*****************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                  */
         /*****************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                   (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         uni_lwr = UniTransLower(locale_object, uni_upr);
         return (ULS_SUCCESS);

}
```

UniTransUpper

UniTransUpper converts Unicode character to uppercase using the specified locale.

Format

```
#include <unidef.h>
```

UniChar UniTransUpper
(const LocaleObject locale_object, UniChar uc)

Parameters

locale_object (const LocaleObject)
: A locale object created by UniCreateLocaleObject or NULL

uc (UniChar)
: The character to be transformed.

Returns

return value (UniChar) - returns
: This function returns the transformed character. The input character is returned if there is no

transformation defined for the character in locale_object.

Related Functions

- UniTransLower

Example

```
This example shows how to convert a Unicode character to uppercase.
#include <stdio.h>

#include <unidef.h>
int main(void) {
LocaleObject  locale_object;
UniChar       uni_lwr = L'c';
UniChar       uni_upr;
int           rc;
         /******************************************************************/
         /* Assumes LANG environment variable set to a valid locale name, */
         /* such as fr_FR                                                  */
         /******************************************************************/
         rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER,
                                   (UniChar *)L"", &locale_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateLocaleObject error: return code = %u\n", rc);
           return 1;
         }
         uni_upr = UniTransUpper(locale_object, uni_lwr);
         return (ULS_SUCCESS);

}
```

Conversion Functions

It is expected that most of the processing by applications using the ULS will be done using UniChar* strings. Yet, many applications will need to export data to and import data from non-UCS encodings (for example, ASCII or EBCDIC). For this purpose, a set of functions are defined to perform conversions between UCS and non-UCS encodings. The Uconv name is used to indicate these functions. The Uconv functions are capable of doing only UCS conversions, such as converting to/from UCS.

UniCreateUconvObject

UniCreateUconvObject creates and initializes a Uconv object.

Format

```
#include <uconv.h>
```

int UniCreateUconvObject
(UniChar *cpname, UconvObject *uconv_object)

Parameters

cpname (UniChar *)
: Name of the UCS conversion.

uconv_object (UconvObject *)
: The conversion object being returned.

Returns

return value (int) - returns
: Upon completion, UniCreateUconvObject returns one of the following:

ULS_SUCCESS
: Conversion object successfully initialized.

ULS_INVALID
: The conversion specified by cpname is not recognized by the implementation.

Remarks

UniCreateUconvObject returns a conversion object that describes a UCS-2 conversion between the code page specified by **cpname** and UCS.

A conversion object remains valid until it is freed.

The cpname field is normally the Unicode string *IBM-* followed by the decimal number of the code page. Other names may be used. UCONV tables are kept in the *\language\codepage* directory on the boot drive.

If the cpname parameter contains an empty string, UniCreateUconvObject will create a conversion object based upon the value of the process codepage setting.

UniCreateUconvObject allows modifiers to be concatenated onto cpname, these modifiers change the default behavior of conversion objects. The caller can concatenate the following modifiers onto cpname.

Modifiers are separated from the conversion object name by an at sign (@), and multiple modifiers are separated by a comma (,).

displaymask

```
    @map=data        All characters less than space are controls.
                     (default)
    @map=display     All characters less than space are glyphs.
    @map=cdra        Use IBM standard control conversion.
    @map=clrf        CR and LF are controls, others are glyphs.
```

converttype

```
    @path=yes        When performing Unicode conversions strings
                     are assumed to contain pathnames. This
                     setting is only applicable when converting
```

```
                 to or from DBCS codepages.
                 (default)
@path=no         When performing Unicode conversions strings
                 are assumed to contain non path data. This
                 setting is only applicable when converting
                 to or from DBCS codepages.
```

endian

```
@endian=Source:Target
@endian=Both
Source applies to UniUconvFromUcs; Target applies to
UniUconvToUcs.  If only one endian is given, it applies
to both source and target.
The endian type can be one of the following:
      system     Use system endian.
      big        Use big endian.
      little     Use little endian.
                 (default)
 For example @endian=little
             @endian=big:system
```

options

```
@sub=yes          Perform substitutions when converting to and
                  from Unicode.
@sub=no           Do not perform substitutions when converting
                  to and from Unicode.
@sub=to-ucs       Only perform substitutions when converting to
                  Unicode.
@sub=from-ucs     Only perform substitutions when converting from
                  Unicode.
                  (default)
@subchar=\xXX     Where XX is a hex number
@subchar=\DD      Where DD is a decimal number
                  The substitution character attribute specifies
                  which character the conversion object should
                  use when there is no identical character for
                  a given code element while converting from
                  Unicode.
@subuni=\xXX\xXX  Where XX is a hex number
@subuni=\xXXXX    Where XXXX is a hex number
                  The substitution character attribute specifies
                  which character the conversion object should
                  use when there is no identical character for
                  a given code element while converting to
                  Unicode.
```

Examples of typical usage:

IBM-942@path=yes,map=display

```
This example creates a conversion object based upon an IBM-942
encoding. When conversions are performed all strings will be treated
as pathnames and all characters less than space will be considered
to be glyphs.
```

@path=yes,sub=no

```
This example creates a conversion object based upon the current
```

```
process codepage setting. When conversions are performed all strings
will be treated as pathnames and no substitutions will occur.
```

IBM-850@path=no,sub=yes

```
This example creates a conversion object based upon an IBM-850
encoding. When conversions are performed all strings will be treated
as non pathnames and substitutions will occur when converting to and
from Unicode if necessary.
```

UniCreateUconvObject returns a conversion object in uconv_object for use in subsequent calls to either UniUconvFromUcs or UniUconvToUcs.

Related Functions

- UniFreeUconvObject
- UniQueryUconvObject
- UniSetUconvObject

Example

```
This example shows how to create a conversion object.
#include <stdio.h>

#include <uconv.h>
int main(void) {
UconvObject  uconv_object = NULL;

int          rc = ULS_SUCCESS;
         /*******************************************************************/
         /* Create a conversion object based upon the process codepage      */
         /* setting with the path modifier set                              */
         /*******************************************************************/
         rc = UniCreateUconvObject((UniChar *)L"@path=yes", &uconv_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateUconvObject error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;

}
```

UniFreeUconvObject

UniFreeUconvObject frees a conversion object.

Format

```
#include <uconv.h>
```

int UniFreeUconvObject
    (UconvObject uconv_object)

Parameters

uconv_object (UconvObject)
: Conversion object created by a call to UniCreateUconvObject.

Returns

return value (int) - returns

Upon completion, UniFreeUconvObject returns one of the following values:

ULS_SUCCESS
All resources associated with **uconv** have been successfully freed.

ULS_BADOBJECT
The uconv_object argument is not a valid conversion object.

Remarks

UniFreeUconvObject closes the conversion object.

Related Functions

- UniCreateUconvObject
- UniQueryUconvObject
- UniSetUconvObject

Example

```
This example shows how to create and free a conversion object.
#include <stdio.h>

#include <uconv.h>
int main(void) {
UconvObject  uconv_object = NULL;

int          rc = ULS_SUCCESS;
         /*********************************************************************/
         /* Create a conversion object based upon the process codepage       */
         /* setting with the path modifier set                               */
         /*********************************************************************/
         rc = UniCreateUconvObject((UniChar *)L"@path=yes", &uconv_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateUconvObject error: return code = %u\n", rc);
           return 1;
         }
         rc = UniFreeUconvObject(uconv_object);
         if (rc != ULS_SUCCESS) {
           printf("UniFreeUconvObject error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;

}
```

UniMapCpToUcsCp

UniMapCpToUcsCp converts a code page number into a code page represented as a UniChar string that is acceptable as input to UniCreateUconvObject.

Format

```
#include <uconv.h>
```

int UniMapCpToUcsCp
(unsigned long ulCodepage, UniChar *ucsCodepage, size_t n)

Parameters

ulCodepage (unsigned long) - input
A code page as returned from DosQueryCp. If the value is zero the current process codepage value will be used in determining the returned Unicode string.

ucsCodepage (UniChar *) - output
A buffer for placing the Unicode string.

n (size_t) - input
Size of the ucsCodepage buffer in Unicode characters. This should be at least 12 Unicode characters.

Returns

retcode (int) - returns
Error code.

UniMapCpToUcsCp returns one of the following values:

ULS_SUCCESS
The code page number was successfully converted to a Unicode string.
ULS_INVALID
An invalid code page number or buffer was passed in; the contents of ucsCodepage are undefined.

Related Functions

- UniMapCtryToLocale

Example

```
This example shows how to convert a codepage number to a Unicode string.
#include <stdio.h>

#include <uconv.h>
int main(void) {
UniChar      ucs_code_page[12];
size_t       num_elems = 12;
UconvObject  uconv_object = NULL;
int          rc = ULS_SUCCESS;
```

```
         /********************************************************************/
         /* Convert a code page number to a unichar string                   */
         /********************************************************************/
```

rc = UniMapCpToUcsCp(850, ucs_code_page, num_elems);

```
         if (rc != ULS_SUCCESS) {
           printf("UniMapCpToUcsCp error: return code = %u\n", rc);
           return 1;
         }
         rc = UniCreateUconvObject(ucs_code_page, &uconv_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateUconvObject error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;
```

}

UniQueryUconvObject

UniQueryUconvObject queries the attributes of a conversion object.

Format

```
#include <uconv.h>
```

int UniQueryUconvObject
(UconvObject uobj, uconv_attribute_t *attr, size_t size, char first[256], char other[256], udcrange_t udcrange[32])

Parameters

uobj (UconvObject) - input
The conversion object created by a call to UniCreateUconvObject.

attr (uconv_attribute_t *) - output
Pointer to uconv_attribute_t; receives attribute information.

size (size_t) - input
Specifies the size of the attribute buffer. This must be at least as large as version 0 of the uconv_attribute_t structure.

first[256] (char) - output
Gives an array of starting bytes for a multibyte character set. For some forms of stateful code pages, the length is based on state and not this table. If this parameter is NULL, no value is returned. Each byte has one of the following values:

1
Valid single byte character.

2

for a double-byte character.

3

for a triple-byte character.

255

code point.

other[256] (char) - output

An array indicating when the byte is used a secondary byte in a multi-byte sequence. This is used to allocate buffers. There are two possible values for each byte:

0

This is not used as a secondary character.

1

This is used as a secondary character.

udcrange[32] (udcrange_t) - output

A set of ranges of characters that make up the user-defined character range.

Returns

retcode (int) - returns

Error code.

UniQueryUconvObject returns one of the following values:

ULS_SUCCESS

The conversion object data was successfully returned

Remarks

UniQueryUconvObject queries the attributes and characteristics of the given conversion object. The attributes are used to modify the default conversion.

The substitution character attributes specify to the conversion object how to perform in cases that there is no identical character for a given code element. UniQueryUconvObject may be used to query the substitution characters used by the conversion.

Some of these are static and bound to the conversion table; others are settable through UniSetUconvObject

The *attr*, *first*, *other*, and *udcrange* parameters can be NULL to indicate that this data should not be returned.

See the uconv_attribute_t to see the conversion object attributes; the structure indicates which fields can be queried and which can be set through UniSetUconvObject.

Related Functions

- UniCreateUconvObject
- UniFreeUconvObject

- UniSetUconvObject

Example

```
This example shows how to query a conversion object.
#include <stdio.h>

#include <uconv.h>
int main(void) {
uconv_attribute_t   attr;
UconvObject         uconv_object = NULL;
int                 rc = ULS_SUCCESS;
         /********************************************************************/
         /* Create a conversion object based upon the process codepage       */
         /* setting with the path modifier set                               */
         /********************************************************************/
         rc = UniCreateUconvObject((UniChar *)L"@path=yes", &uconv_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateUconvObject error: return code = %u\n", rc);
           return 1;
         }
         /* Query the conversion object */
         rc = UniQueryUconvObject(uconv_object, &attr,
                                  sizeof(uconv_attribute_t), NULL,
                                  NULL, NULL);
         if (rc != ULS_SUCCESS) {
           printf("UniQueryUconvObject error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;

}
```

UniSetUconvObject

UniSetUconvObject sets the attributes of a conversion object.

Format

```
#include <uconv.h>
```

int UniSetUconvObject
(UconvObject uconv_object, uconv_attribute_t *attr_t)

Parameters

uconv_object (UconvObject) - input
    The conversion object created by a call to UniCreateUconvObject. :evallist. :eparm.

attr_t (uconv_attribute_t *) - input
    The uconv_attribute_t structure, which the caller sets with values of the conversion object attributes. The caller can set these fields:

options
Substitution options, which can have one of these values:

UCONV_OPTION_SUBSTITUTION_FROM_UNICODE
UCONV_OPTION_SUBSTITUTION_TO_UNICODE
UCONV_OPTION_SUBSTITUTION_BOTH

endian
Source and target endian. Thsi is a structure containing a source and target endian field. Source applies to UniUconvFromUcs; target applies to UniUconvToUcs. Each of the fields can contain one of the following values:

0x000
Use system endian.

0xfeff
Use big endian.

0xfffe
Use little endian.

displaymask
A 32-bit display and data mask. Each bit represents a control character below space (1<<char). If the bit is 0, the character is treated as a display glyph. If the bit is 1, the character is treated as a control. There are several predefined values for this mask, but any value can be used:

DSPMASK_DATA
All characters less than space are controls.

DSPMASK_DISPLAY
All characters less than space are glyphs.

DSPMASK_CRLF
CR and LF are controls: Others are glyphs.

converttype
Conversion type. This is a set of flags. The following flags exist and can be ORed together:

CVTTYPE_CTRL7F
Treat the 0x7f character as a control.

CVTTYPE_CDRA
Use IBM standard control conversion. If this bit is not set, controls are converted to an equal value. Some conversions always do control conversions.

CVTTYPE_PATH
When performing Unicode conversions strings are assumed to contain pathnames. This setting is only applicable when converting to or from DBCS codepages.

subchar_len
Code page substitution length. This can be a value between 1 and 13 to indicate the substitution length. It may not exceed the maximum size character in the encoding. A value of zero indicates that the substitution character from the conversion table should be used.

subchar
Substitution bytes. This is the actual value whose length is specified by subchar_len.

subuni_len
Unicode substitution length. This can be either 0 or 1. A zero indicates that the Unicode substitution from the conversion table should be used.

subuni
If subuni_len is set to 1, the first element in this array gives the Unicode substitution character.

state
When the state is set to 0, the conversion object is put into the base conversion state.

Returns

return value (int) - returns
Return Codes
ULS_SUCCESS
Indicates success.
ULS_BADATTR
Indicates an invalid parameter.

Remarks

UniSetUconvObject sets the attributes of the given conversion object. The attributes are used to modify the default conversion. It is left up to each conversion to decide which attributes it will recognize.

The substitution character attributes specify to the conversion object how to perform in cases that there is no identical character for a given code element.

Remarks

This example sets the displaymask to all display and path to yes, meaning all code points below space are mapped as glyphs and not as controls. It also treats data as pathnames. To modify only some attributes, a query should first be done using UniQueryUconvObject.

Related Functions

- UniCreateUconvObject
- UniFreeUconvObject
- UniQueryUconvObject

Example

```
This example shows how to set a conversion object.
#include <stdio.h>

#include <uconv.h>
int main(void) {
uconv_attribute_t   attr;
UconvObject         uconv_object = NULL;
int                 rc = ULS_SUCCESS;
         /*****************************************************************/
         /* Create a conversion object based upon the process codepage       */
         /* setting with the path modifier set                               */
         /*****************************************************************/
         rc = UniCreateUconvObject((UniChar *)L"@path=yes", &uconv_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateUconvObject error: return code = %u\n", rc);
           return 1;
         }
         /* Query the conversion object */
         rc = UniQueryUconvObject(uconv_object, &attr,
                                  sizeof(uconv_attribute_t), NULL,
                                 NULL, NULL);
         if (rc != ULS_SUCCESS) {
           printf("UniQueryUconvObject error: return code = %u\n", rc);
           return 1;
         }
         /* Turn the path modifier and display attributes on */
         attr.converttype = attr.converttype | CVTTYPE_PATH;
         attr.displaymask = DSPMASK_DISPLAY;
         rc = UniSetUconvObject(uconv_object, &attr);
         if (rc != ULS_SUCCESS) {
           printf("UniSetUconvObject error: return code = %u\n", rc);
           return 1;
         }
         return ULS_SUCCESS;

}
```

UniUconvFromUcs

UniUconvFromUcs converts UCS characters to code page characters.

Format

```
#include <uconv.h>
```

int UniUconvFromUcs
(UconvObject uconv_object, UniChar **ucsbuf, size_t *UniCharsleft, void **outbuf, size_t *outbytesleft, size_t *nonidentical)

Parameters

uconv_object (UconvObject)
: Conversion object created by a call to UniCreateUconvObject.

ucsbuf (UniChar **)
: Input buffer.

UniCharsleft (size_t *)
: Number of UniChar elements in ucsbuf.

outbuf (void **)
: Output buffer.

outbytesleft (size_t *)
: Size of outbuf, in units of bytes.

nonidentical (size_t *)
: Number of nonidentical conversions.

Returns

return value (int) - returns

UniUconvToUcs returns one of the following values:

ULS_SUCCESS

Conversion successful.

ULS_BUFFERFULL

Input conversion stopped due to lack of space in the output buffer.

ULS_ILLEGALSEQUENCE

Input conversion stopped due to an incomplete character or shift sequence at the end of the input buffer. This condition will be given when the last UniChar element is recognized as a combining character.

ULS_INVALID

The uconv_object argument is not a valid, open conversion object.

UniUconvFromUcs updates the variables pointed to by the arguments to reflect the extent of the conversion and returns, in **nonidentical**, the number of substitution (nonidentical) conversions performed. If the entire string in the input buffer is converted, the value pointed to by **UniCharsleft** will be 0. If the input conversion is stopped due to any condition mentioned above, the value pointed to by **UniCharsleft** will be nonzero. If any error occurs, UniUconvToUcs returns a nonzero value.

Remarks

UniUconvFromUcs converts a sequence of code elements, in the array specified by **ucsbuf**, into a sequence of corresponding characters in another code page, in the array specified by **outbuf**. The code page of the **outbuf** is the string specified in the UniCreateUconvObject call that returned the conversion object, **uconv_object**. The **ucsbuf** argument points to a variable that points to the first UniChar in the input buffer, and the **UniCharsleft** indicates the number of UniChar elements to the end of the buffer to be converted. The **outbuf** argument points to a variable that points to the first available character in **outbuf**, and **outbytesleft** indicates the number of bytes available to the end of the buffer.

If the **outbuf** buffer is not large enough to hold the entire converted input, conversion stops just prior to the input UniChar that would cause the output buffer to overflow. The variable pointed to by **ucsbuf** is updated to point to the UniChar following the last UniChar successfully used in the conversion. The value pointed to by **outbytesleft** is decremented to reflect the number of bytes still available in **outbuf**.

If UniUconvFromUcs encounters a code element in the **ucsbuf** that is legal, but for which an identical character does not exist in the target code page, UniUconvFromUcs replaces the character with a predefined substitution character, if the attributes of the conversion object allow this operation. If substitution is not selected, an error is returned and conversion stops after the previous successfully converted UniChar.

Related Functions

- UniUconvToUcs
- UniStrFromUcs
- UniStrToUcs

Example

```
This example shows how to convert a Unicode string to code page characters.
#include <stdio.h>
```

```
#include <uconv.h>
int main(void) {
UconvObject  uconv_object = NULL;
int          rc = ULS_SUCCESS;
size_t       out_bytes_left;
size_t       uni_chars_left;
size_t       num_subs;
size_t       char_buf_size = 50;
char         char_buffer[50];
char         *pout_char_str;
UniChar      *pin_uni_str;
UniChar      uni_data[] = L"UniCode string to convert";
         /********************************************************************/
         /* Create a conversion object based upon the process codepage       */
         /********************************************************************/
         rc = UniCreateUconvObject((UniChar *)L"", &uconv_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateUconvObject error: return code = %u\n", rc);
           return 1;
         }
        /*********************************************************************/
        /* pin_uni_str points to the unicode string to be converted to       */
        /* codepage characters                                               */
        /*********************************************************************/
        pout_char_str = char_buffer;
        pin_uni_str = uni_data;
        uni_chars_left = UniStrlen(pin_uni_str)+1;
        out_bytes_left = char_buf_size;
        /*********************************************************************/
        /* make call to convert unicode string  to codepage characters       */
        /*********************************************************************/
        rc = UniUconvFromUcs(uconv_object, &pin_uni_str, &uni_chars_left,
                            (void **)&pout_char_str, &out_bytes_left,
                            &num_subs);
        if(rc != ULS_SUCCESS && uni_chars_left > 0) {
           printf("UniUconvFromUcs error: return code = %u\n", rc);
           printf("Unicode string was not completely converted\n");
           return 1;
        }
```

return ULS_SUCCESS;
}

UniUconvToUcs

UniUconvToUcs converts a code page string to a UCS string.

Format

```
#include <uconv.h>
```

int UniUconvToUcs
(UconvObject uconv_object, void **inbuf, size_t *inbytesleft, UniChar **ucsbuf, size_t *UniCharsleft, size_t *nonidentical)

Parameters

uconv_object (UconvObject)
: Conversion object created by a call to UniCreateUconvObject.

inbuf (void **)
: Input buffer.

inbytesleft (size_t *)
: Size of inbuf, in units of bytes.

ucsbuf (UniChar **)
: Output buffer.

UniCharsleft (size_t *)
: Number of Unichar elements in ucsbuf.

nonidentical (size_t *)
: Number of non-identical conversions.

Returns

return value (int) - returns

UniUconvToUcs returns one of the following values:

ULS_SUCCESS
: Conversion successful.

ULS_INVALID
: Input conversion stopped due to an error condition such as lack of buffer space.

UniUconvToUcs updates the variables pointed to by the arguments to reflect the extent of the conversion and returns, in **nonidentical**, the number of substitutions (non-identical) conversions performed. If the entire string in the input buffer is converted, the value pointed to by **inbytesleft** will be zero. If the input conversion is stopped due to any condition mentioned above, the value pointed to by **inbytesleft** will be non zero and a non zero value is returned to indicate the condition. If an error occurs, UniUconvToUcs returns a non zero value.

Remarks

UniUconvToUcs converts a sequence of characters encoded in one code page, in the array specified by **inbuf**, into a sequence of corresponding UCS code elements, in the array specified by **ucsbuf**. The code page of the **inbuf** is the string specified in the UniCreateUconvObject call that returned the conversion object, **uconv_object**. The **inbuf** argument points to a variable that points to the first byte in the input buffer, and the **inbytesleft** indicates the number of bytes to the end of the buffer to be converted. The **ucsbuf** argument points to a variable that points to the first available UniChar in **ucsbuf**, and **UniCharsleft** indicates the number of UniChar elements available to the end of the buffer.

If a sequence of bytes within **inbuf** does not form a valid character in the specified code page and substitution to UCS is not turned on, conversion stops after the previous successfully converted character. If the input buffer ends with an incomplete character, conversion stops after the previous successfully converted bytes. If the **ucsbuf** buffer is not large enough to hold the entire converted input, conversion stops just prior to the input bytes that would cause the output buffer to overflow. The variable pointed to by **inbuf** is updated to point to the byte following the last byte successfully used in the conversion. The value pointed to by **UniCharsleft** is decremented to reflect the number of UniChar elements still available in **ucsbuf**.

Related Functions

- UniUconvFromUcs
- UniStrFromUcs
- UniStrToUcs

Example

```
This example shows how to convert code page encoded characters to Unicode.
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <uconv.h>
int main(void) {
UconvObject  uconv_object = NULL;
int          rc = ULS_SUCCESS;
size_t       in_bytes_left;
size_t       uni_chars_left;
size_t       num_subs;
int          uni_buf_length = 50;
UniChar      uni_buffer[50];
UniChar      *pout_uni_str;
char         char_data[] = "Character string to convert";
char         *pin_char_str;
         /********************************************************************/
         /* Create a conversion object based upon the process codepage      */
         /********************************************************************/
         rc = UniCreateUconvObject((UniChar *)L"", &uconv_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateUconvObject error: return code = %u\n", rc);
           return 1;
         }
        /*********************************************************************/
        /* pin_char_str points to the character string to be converted to    */
        /* Unicode characters                                                */
        /*********************************************************************/
        pout_uni_str = uni_buffer;
        pin_char_str = char_data;
        uni_chars_left = uni_buf_length;
        in_bytes_left = ( strlen(char_data) + 1 ) * sizeof(char);
       /**********************************************************************/
       /* make call to convert codepage character string to a Unicode string */
       /**********************************************************************/
       rc = UniUconvToUcs(uconv_object, (void **)&pin_char_str, &in_bytes_left,
                          &pout_uni_str, &uni_chars_left,
                          &num_subs);
        if(rc != ULS_SUCCESS && in_bytes_left > 0) {
           printf("UniUconvToUcs error: return code = %u\n", rc);
           printf("Character string was not completely converted to Unicode\n");
           return 1;
```

```
          }

    return ULS_SUCCESS;
    }
```

UniStrFromUcs

UniStrFromUcs converts a UCS string to a code page string.

Format

```
#include <uconv.h>
```

int UniStrFromUcs
    (UconvObject uconv_object, char *outbuf, UniChar *ucsstr, int size)

Parameters

uconv_object (UconvObject)
    Conversion object created by a call to UniCreateUconvObject.

outbuf (char *)
    Output buffer to hold converted string.

ucsstr (UniChar *)
    Null terminated Unicode string.

size (int)
    number of bytes that output buffer can hold

Returns

return value (int) - returns
    UniStrFromUcs returns one of the following values:
    ULS_SUCCESS
        Conversion successful.
    ULS_BADOBJECT
        The **uconv_object** argument is not a valid, open conversion object.
    ULS_BUFFERFULL
        Input conversion stopped due to lack of space in the output buffer.
    ULS_ILLEGALSEQUENCE
        Input conversion stopped due to an input byte that does not belong to the input code page.

    UniStrFromUcs always performs conversions with substitution on.

Remarks

UniStrFromUcs converts a sequence of code elements up to and including the null terminator, in the array specified by **ucsstr**, into a sequence of corresponding characters in another code page in the array specified by **outbuf**. The code page of **outbuf** is the string specified in the UniCreateUconvObject call that returned the conversion object, **uconv_object**. The **ucsstr** argument points to a variable that points to the first UniChar in the input bufer. The **outbuf** argument points to a variable that points to the first available character in **outbuf**, and **size** indicates the number of bytes available to the end of the buffer.

If the **outbuf** buffer is not large enough to hold the entire converted input, conversion stops just prior to the input UniChar that would cause the output buffer to overflow.

If UniStrFromUcs encounters a code element in the **ucsstr** that is legal, but for which an identical character does not exist in the target code page, UniStrFromUcs replaces the character with a predefined substitution character.

Related Functions

- UniUconvFromUcs
- UniUconvToUcs
- UniStrToUcs

Example

```
This example shows how to convert a Unicode string to code page characters.
#include <stdio.h>

#include <uconv.h>
int main(void) {
UconvObject  uconv_object = NULL;
int          rc = ULS_SUCCESS;
size_t       buf_size = 50;
char         char_buffer[50];
UniChar      uni_data[] = L"UniCode string to convert";
         /*******************************************************************/
         /* Create a conversion object based upon the process codepage      */
         /*******************************************************************/
         rc = UniCreateUconvObject((UniChar *)L"", &uconv_object);
         if (rc != ULS_SUCCESS) {
           printf("UniCreateUconvObject error: return code = %u\n", rc);
           return 1;
         }

        /********************************************************************/
        /* make call to convert unicode string  to codepage characters      */
        /********************************************************************/
        rc = UniStrFromUcs(uconv_object, char_buffer, uni_data, buf_size);
        if(rc != ULS_SUCCESS) {
           printf("UniStrFromUcs error: return code = %u\n", rc);
           printf("Unicode string was not completely converted\n");
           return 1;
        }
```

return ULS_SUCCESS;
}

UniStrToUcs

UniStrToUcs converts a code page string to a UCS string.

Format

```
#include <uconv.h>
```

int UniStrToUcs
(UconvObject uconv_object, void **inbuf, size_t *inbytesleft, UniChar **ucsbuf, size_t *UniCharsleft, size_t *nonidentical)

Parameters

uconv_object (UconvObject)
Conversion object created by a call to UniCreateUconvObject.

inbuf (void **)
Input buffer.

inbytesleft (size_t *)
Size of inbuf, in units of bytes.

ucsbuf (UniChar **)
Output buffer.

UniCharsleft (size_t *)
Number of Unichar elements in ucsbuf.

nonidentical (size_t *)
Number of non-identical conversions.

Returns

return value (int) - returns
UniUconvToUcs returns one of the following values:

ULS_SUCCESS
Conversion successful.

ULS_BADOBJECT
The **uconv_object** argument is not a valid, open conversion object.

ULS_BUFFERFULL
Input conversion stopped due to lack of space in the output buffer.

ULS_ILLEGALSEQUENCE
Input conversion stopped due to an input byte that does not belong to the input code page.

ULS_INVALID
Input conversion stopped due to an incomplete character or shift sequence at the end of the input buffer.

UniUconvToUcs updates the variables pointed to by the arguments to reflect the extent of the conversion and returns, in **nonidentical**, the number of substitutions (non-identical) conversions performed. If the entire string in the input buffer is converted, the value pointed to by **inbytesleft** will be zero. If the input conversion is stopped due to any condition mentioned above, the value pointed to by **inbytesleft** will be non zero and a non zero value is returned to indicate the condition. If an error occurs, UniUconvToUcs returns a non zero value.

Remarks

UniUconvToUcs converts a sequence of characters encoded in one code page, in the array specified by **inbuf**, into a sequence of corresponding UCS code elements, in the array specified by **ucsbuf**. The code page of the **inbuf** is the string specified in the UniCreateUconvObject call that returned the conversion object, **uconv_object**. The **inbuf** argument points to a variable that points to the first byte in the input buffer, and the **inbytesleft** indicates the number of bytes to the end of the buffer to be converted. The **ucsbuf** argument points to a variable that points to the first available UniChar in **ucsbuf**, and **UniCharsleft** indicates the number of UniChar elements available to the end of the buffer.

If a sequence of bytes within **inbuf** does not form a valid character in the specified code page and substitution to UCS is not turned on, conversion stops after the previous successfully converted character. If the input buffer ends with an incomplete character, conversion stops after the previous successfully converted bytes. If the **ucsbuf** buffer is not large enough to hold the entire converted input, conversion stops just prior to the input bytes that would cause the output buffer to overflow. The variable pointed to by **inbuf** is updated to point to the byte following the last byte successfully used in the conversion. The value pointed to by **UniCharsleft** is decremented to reflect the number of UniChar elements still available in **ucsbuf**.

Related Functions

- UniUconvFromUcs
- UniUconvToUcs
- UniStrFromUcs

Example

```
This example shows how to convert code page encoded characters to Unicode.
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <uconv.h>
int main(void) {
UconvObject  uconv_object = NULL;
int          rc = ULS_SUCCESS;
size_t       in_bytes_left;
size_t       uni_chars_left;
size_t       num_subs;
int          uni_buf_length = 50;
UniChar      uni_buffer[50];
UniChar      *pout_uni_str;
char         char_data[] = "Character string to convert";
char         *pin_char_str;
         /*********************************************************************/
```

```
          /* Create a conversion object based upon the process codepage       */
          /*********************************************************************/
          rc = UniCreateUconvObject((UniChar *)L"", &uconv_object);
          if (rc != ULS_SUCCESS) {
            printf("UniCreateUconvObject error: return code = %u\n", rc);
            return 1;
          }
         /**********************************************************************/
         /* pin_char_str points to the character string to be converted to    */
         /* Unicode characters                                                 */
         /**********************************************************************/
         pout_uni_str = uni_buffer;
         pin_char_str = char_data;
         uni_chars_left = uni_buf_length;
         in_bytes_left = ( strlen(char_data) + 1 ) * sizeof(char);
        /***********************************************************************/
        /* make call to convert codepage character string to a Unicode string */
        /***********************************************************************/
        rc = UniUconvToUcs(uconv_object, (void **)&pin_char_str, &in_bytes_left,
                          &pout_uni_str, &uni_chars_left,
                          &num_subs);
         if(rc != ULS_SUCCESS && in_bytes_left > 0) {
            printf("UniUconvToUcs error: return code = %u\n", rc);
            printf("Character string was not completely converted to Unicode\n");
            return 1;
         }
```

return ULS_SUCCESS;
}

ULS Data Types

The following data types are used by the Unicode functions.

AttrObject

AttrObject is used to determine character classifications.

```
typedef void *AttrObject;
```

---

conv_endian_t

conv_endian_t Information about the source and target endian.

```
typedef struct _conv_endian_rec {
  unsigned short       source;
  unsigned short       target;
} conv_endian_t;
```

Parameters

source (unsigned short)
: Source information.

target (unsigned short)
: Target information.

LocaleItem

LocaleItem is used to identify a language or cultural item within a locale.

```
typedef int LocaleItem;
```

LocaleObject

LocaleObject is used by APIs that require language or cultural sensitive processing.

```
typedef void *LocaleObject;
```

LocaleToken

LocaleToken is used as a shorthand method for identifying locales.

```
typedef unsigned int LocaleToken;
```

struct UniLconv

struct UniLconv describes the locale conventions.

```
struct UniLconv {
  UniChar *decimal_point;
  UniChar *thousands_sep;
  short        *grouping;
  UniChar *int_curr_symbol;
  UniChar *currency_symbol;
  UniChar *mon_decimal_point;
  UniChar *mon_thousands_sep;
  short        *mon_grouping;
  UniChar *positive_sign;
  UniChar *negative_sign;
  short        int_frac_digits;
  short        frac_digits;
  short        p_cs_precedes;
  short        p_sep_by_space;
  short        n_cs_precedes;
  short        n_sep_by_space;
  short        p_sign_posn;
  short        n_sign_posn;
  short        os2_mondecpt;
  UniChar *debit_sign;
  UniChar *credit_sign;
  UniChar *left_parenthesis;
  UniChar *right_parenthesis;
};
```

Parameters

decimal_point (UniChar *)
: Non-monetary decimal point.

thousands_sep (UniChar *)
: Non-monetary thousands separator.

grouping (short *)
: Size of each group of digits in non-monetary quantities.

int_curr_symbol (UniChar *)
: International currency symbol and separator.

currency_symbol (UniChar *)
: Local currency symbol.

mon_decimal_point (UniChar *)
: Monetary decimal point.

mon_thousands_sep (UniChar *)
: Monetary thousands separator.

mon_grouping (short *)
: Size of each group of digits in monetary quantities.

positive_sign (UniChar *)
: Non-negative values sign.

negative_sign (UniChar *)
: Negative values sign.

int_frac_digits (short)
: Number of fractional digits for international currency.

frac_digits (short)
: Number of fractional digits for local currency.

p_cs_precedes (short)
: Nonnegative currency symbol 1-precedes, 0-succeeds.

p_sep_by_space (short)
: Nonnegative currency symbol 1-space, 0-no space.

n_cs_precedes (short)
: Negative currency symbol 1-precedes, 0-succeeds.

n_sep_by_space (short)
: Negative currency symbol 1-space, 0-no space.

p_sign_posn (short)
: Positioning of nonnegative monetary sign.

n_sign_posn (short)
: Positioning of negative monetary sign.

os2_mondecpt (short)
: OS2 currency symbol positioning.

debit_sign (UniChar *)
: Non-negative valued debit monetary symbol.

credit_sign (UniChar *)
: Negative valued credit monetary symbol.

left_parenthesis (UniChar *)
: Negative valued left parenthesis monetary symbol.

right_parenthesis (UniChar *)
: Negative valued right parenthesis monetary symbol.

---

uconv_attribute_t

uconv_attribute_t This structure describes the attributes and characteristics of a conversion object. All of these fields are queryable through UniQueryUconvObject. Some of the fields are settable through UniSetUconvObject; these are marked in the descriptions.

```
typedef struct _uconv_attribute_t {
  unsigned long  version;
  char           mb_min_len;
  char           mb_max_len;
  char           usc_min_len;
  char           usc_max_len;
  unsigned short esid;
  char           options;
  char           state;
  conv_endian_t  endian;
  unsigned long  displaymask;
  unsigned long  converttype;
  unsigned short subchar_len;
  unsigned short subuni_len;
  char           subchar[16];
  UniChar        subuni[8];
} uconv_attribute_t;
typedef uconv_attribute_t *uconv_attribute_t;
```

Parameters

version (unsigned long)
    Version (must be zero). Settable.
mb_min_len (char)
    Minimum character size.
mb_max_len (char)
    Maximum character size.
usc_min_len (char)
    UCS minimum character size.
usc_max_len (char)
    UCS maximum character size.
esid (unsigned short)
    Encoding scheme ID.
options (char)
    Substitution options. Settable.
state (char)
    Current state. Settable.
endian (conv_endian_t)
    Source and target chain. Settable.
displaymask (unsigned long)
    Display and data mask. Settable.
converttype (unsigned long)
    Conversion type. Settable.
subchar_len (unsigned short)
    MBCS sub-character length. Settable.
subuni_len (unsigned short)
    Unicode sub-character length. Settable.
subchar[16] (char)
    MBCS sub-characters. Settable.
subuni[8] (UniChar)
    Unicode sub-characters. Settable.

UconvObject

UconvObject is used by APIs that convert to and from UniCode.

```
typedef void *UconvObject;
```

UNICTYPE

The UNICTYPE structure provides a range of information regarding the type of a character.

```
typedef struct {
  USHORT itype;
  CHAR bidi;
  CHAR charset;
  USHORT extend;
  USHORT codepage;
} UNICTYPE;
```

Parameters

itype (USHORT)
    XPG/4 type attributes (CTYPE1).

bidi (CHAR)
    BiDi type attributes (CTYPE2).

charset (CHAR)
    Character set (CHARSET).

extend (USHORT)
    Win32 Extended attributes (CTYPE3).

codepage (USHORT)
    Codepage bits.

udcrange_t

udcrange_t provides a set of ranges of characters that make up the user-defined character range.

```
typedef struct {
  unsigned short     first;
  unsigned short     last;
} udcrange_t;
```

Parameters

first (unsigned short)
: First code point.

last (unsigned short)
: Last code point.

---

ulsBool

ulsBool

```
typedef int ulsBool;

  0 - FALSE

  1 - TRUE
```

UniChar

A unicode character. Unicode is code that is independent of language and culture, and supports multiple simultaneous character sets.

```
typedef unsigned short UniChar;
```

XformObject

XformObject is used to perform string transformations.

```
typedef void *XformObject;
```

Notices

Unicode Functions (OS/2 Warp)

Copyright Notices

Disclaimers